



HOEPLI  
TECNICA  
PER LA SCUOLA

Paolo Camagni  
Riccardo Nikolassy

# TECNOLOGIE E PROGETTAZIONE DI SISTEMI INFORMATICI E DI TELECOMUNICAZIONI

Nuova Edizione **OPENSCHOOL**

Per l'articolazione INFORMATICA  
degli Istituti Tecnici  
settore Tecnologico

Edizione **OPENSCHOOL**

- |   |               |
|---|---------------|
| 1 | LIBRODITESTO  |
| 2 | E-BOOK+       |
| 3 | RISORSEONLINE |
| 4 | PIATTAFORMA   |

**HOEPLI**



**PAOLO CAMAGNI      RICCARDO NIKOLASSY**

# **Tecnologie e progettazione di sistemi informatici e di telecomunicazioni**

Nuova Edizione OPEN SCHOOL

**Per l'articolazione Informatica  
degli Istituti Tecnici settore Tecnologico**

**VOLUME 1**



**EDITORE ULRICO HOEPLI MILANO**

**Copyright © Ulrico Hoepli Editore S.p.A. 2015**

Via Hoepli 5, 20121 Milano (Italy)

tel. +39 02 864871 – fax +39 02 8052886

e-mail [hoepli@hoepli.it](mailto:hoepli@hoepli.it)

**[www.hoepli.it](http://www.hoepli.it)**



Tutti i diritti sono riservati a norma di legge  
e a norma delle convenzioni internazionali



# Presentazione

Scopo dell'opera è fornire le **basi teoriche e pratiche** per la programmazione **imperativa e ad oggetti**. La **Nuova Edizione Openschool**, concepita secondo le recenti indicazioni ministeriali, è caratterizzata dall'integrazione tra testo cartaceo, libro digitale e risorse digitali a esso collegate (eBook+), ulteriori materiali multimediali disponibili al sito [www.hoepliscuola.it](http://www.hoepliscuola.it) e, a discrezione del docente, la piattaforma didattica.

Completamente revisionato, il testo recepisce le indicazioni ricevute dai docenti che hanno in uso la precedente edizione; in particolare **il testo è stato integrato con le lezioni di laboratorio**, dato che in molti istituti alla materia sono state assegnate non solo ore di teoria.

Il volume mantiene la strutturazione della precedente edizione ed è idealmente organizzato in **tre sezioni**:

- ▶ la **rappresentazione delle informazioni in binario: conversione, codici e codifica dei numeri**;
- ▶ il **sistema operativo**;
- ▶ la **documentazione di un progetto**.

Ogni sezione è composta da unità di apprendimento suddivise in più lezioni. Ciascuna lezione ha una **struttura innovativa** e cerca di essere una reale guida per l'apprendimento; essa, infatti, risulta **essenziale nei contenuti ma ricca di esempi** e di procedure guidate: **per ogni esempio vengono proposte soluzioni guidate passo passo**.

## Metodologia e strumenti didattici

Le finalità e i contenuti dei diversi argomenti affrontati sono descritti dagli **obiettivi generali** e dalle indicazioni **In questa lezione impareremo**; alla fine di ogni lezione, per lo studente sono presenti **esercizi**, anche **interattivi**, di **valutazione delle conoscenze e delle competenze** raggiunte, suddivisi in **domande a risposta multipla, a completamento, esercizi con procedure guidate**.

## Caratteristiche della nuova edizione

La Nuova Edizione Openschool consente di:

- ▶ **scaricare gratuitamente il libro digitale arricchito (eBook+)** che permette in particolare di:
  - ◀ eseguire tutte le **esercitazioni** a risposta chiusa in modo **interattivo**;
  - ◀ accedere alle **gallerie di immagini**;
  - ◀ scaricare gli **approfondimenti tematici, le lezioni e le unità integrative**;
- ▶ disporre di ulteriori esercitazioni online utilizzabili a discrezione del docente per classi virtuali gestibili attraverso la **piattaforma didattica**.

## Aspetti caratterizzanti

- Testo pienamente in linea con le recenti indicazioni ministeriali in merito alle nuove **caratteristiche tecniche e tecnologiche dei libri misti e digitali** e al loro stretto coordinamento con la **piattaforma didattica**.
- Totale **duttilità di utilizzo in funzione delle scelte didattiche o dotazioni tecnologiche**:
  - il libro cartaceo consente di svolgere lezioni complete e attività di laboratorio;
  - l'eBook+, le risorse online e la piattaforma offrono il pieno supporto per una didattica multimediale, a discrezione delle scelte del docente.
- **Lezioni autoconclusive** ricche di esempi ed esercizi, adatte a essere svolte in una lezione o al massimo due.
- **Teoria ridotta al minimo per privilegiare l'aspetto pratico.**

## Materiali online hoepliscuola.it

Sul sito [www.hoepliscuola.it](http://www.hoepliscuola.it) sono disponibili numerose risorse online. In particolare, per lo studente: **approfondimenti**, utili integrazioni del testo e un numero elevato di **esercizi** sia per il **recupero e il rinforzo sia per l'approfondimento** degli argomenti trattati. Per il docente, una sezione riservata presenta alcune **unità didattiche per l'approfondimento** delle tematiche affrontate e un insieme di **schede aggiuntive** per la verifica dei livelli di apprendimento degli studenti, nonché **lezioni** (sotto forma di presentazioni in PowerPoint), **utilizzabili** efficacemente **anche con le LIM**. Materiali ed esercizi possono essere usati anche per creare attività didattiche fruibili tramite la piattaforma didattica accessibile dal sito.

# Struttura dell'opera

The first page shows the 'UNITÀ DI APPRENDIMENTO' (Learning Unit) 1 titled 'Rappresentazione delle informazioni'. It includes a table of contents with topics like 'Sistemi di rappresentazione dei dati', 'Rappresentazione binaria', 'Rappresentazione decimale', 'Rappresentazione di base ottica', and 'Trasformazione dei dati'.

The second page shows 'LEZIONE 1' titled 'Comunicare con il calcolo'. It features a section 'In questo capitolo imparerai...' with points: 'Avere a disposizione strumenti per la comunicazione', 'Comunicare con il calcolo', and 'Utilizzare i calcoli per la comunicazione'.

The third page shows a detailed section on 'Analisi dei dati'. It includes a sub-section 'Analisi dei dati: esempio'. The page also features three small images of different types of data analysis: 'Analisi di dati', 'Analisi di dati', and 'Analisi di dati'.

## ZOOM SU...

Piccole sezioni di approfondimento

This page is titled 'OSSERVAZIONI'. It contains several boxes with text and small icons. One box is titled 'Osservazione 1' and discusses the representation of numbers in binary and decimal systems. Another box is titled 'Osservazione 2' and discusses the representation of numbers in octal and decimal systems.

## OSSERVAZIONI

Un aiuto per comprendere e approfondire

This page is titled 'DEFINIZIONI'. It contains several boxes with text and small icons. One box is titled 'Definizione 1' and discusses the representation of numbers in binary and decimal systems. Another box is titled 'Definizione 2' and discusses the representation of numbers in octal and decimal systems.

## DEFINIZIONI

Spiegazione delle proprietà essenziali dei principali termini, funzioni e concetti trattati nel testo

## PROVA ADESSO!

Per mettere in pratica, in itinere, quanto appreso nella lezione

This page is titled 'ESERCIZI DI LABORATORIO 5' and 'CONVERSIONE ALLA BASE DECIMALE'. It contains several boxes with text and small icons. One box is titled 'Conversione delle diverse basi e decimali: 5 esercizi' and discusses the conversion of different bases to decimal. Another box is titled 'Conversione di numeri binari' and discusses the conversion of binary numbers.

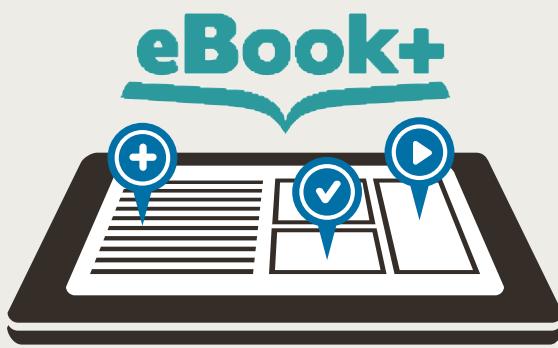
## LABORATORI

Per il rafforzamento dei concetti assimilati, attraverso esercitazioni operative

## ESERCIZI

Ampia sezione di esercizi per la verifica delle conoscenze e delle competenze

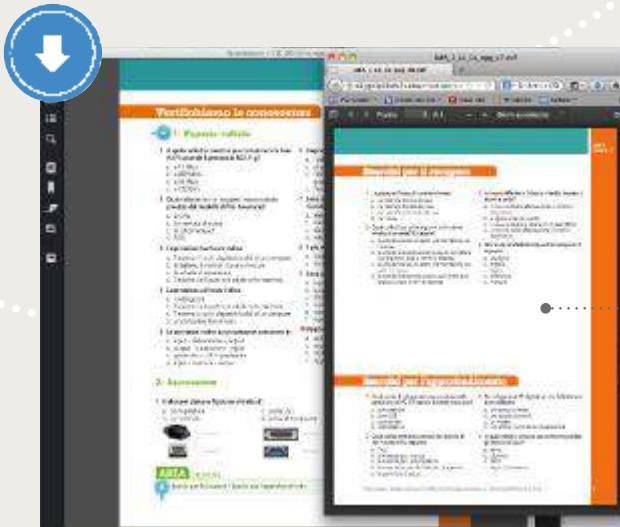
This page is titled 'VERIFICA CON LE ESERCITAZIONI' and 'Verifica con le esercitazioni'. It contains several boxes with text and small icons. One box is titled 'Verifica con le esercitazioni' and discusses the verification of knowledge and skills. Another box is titled 'Verifica con le esercitazioni' and discusses the verification of knowledge and skills.



L'eBook+ riproduce le pagine del libro di testo in versione **digitale e interattiva**. È utilizzabile su tablet, LIM e computer e consente di **annotare, sottolineare ed evidenziare** il testo, salvando il proprio lavoro per poterlo consultare e sincronizzare sui diversi dispositivi. Apposite icone attivano i **contributi digitali integrativi**.

## APPROFONDIMENTI

Contenuti, lezioni e unità integrative



## VIDEO

Video tutorial per esemplificare azioni e procedimenti



## ESERCIZI AGGIUNTIVI

Esercizi per il recupero e l'approfondimento





## ESERCIZI

Esercizi interattivi di varia tipologia con funzione di autocorrezione

The screenshot shows a computer screen with a web-based exercise interface. At the top, there's a navigation bar with the text "Unità di Apprendimento 1 - Il computer". Below it, a large orange sidebar on the left contains sections like "Verifiche e esercizi", "Esercizi interattivi", and "Gallerie di immagini". The main content area displays a multiple-choice question titled "1 - Risposta singola" with four options: A) Super Disk, B) ZIP Disk, C) hard disk drive, and D) memoria ROM. Below the question, there's a section for "Feedback" with the text "Nessuna risposta corretta" and a "Rispondi" button.



## IMMAGINI E GALLERIE DI IMMAGINI

Per esemplificare e rappresentare visivamente i contenuti



## LINK

Rimandi interni  
al volume per navigare  
agevolmente  
tra i contenuti

The screenshot shows a page with a sidebar on the left containing sections like "Verifiche e esercizi", "Esercizi interattivi", and "Gallerie di immagini". The main content area has a heading "1 - Esercizi" and a text block about memory. Below the text, there's a section titled "Esempio di memoria" with a link "Vai all'esempio". The bottom of the page features a "Feedback" section with a checkmark and the text "Esercizio corretto! Trovi altre informazioni su questo argomento nel capitolo 1.2.1.1. - Memoria (vedi pagine 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 5610, 5611, 5612, 5613, 5614, 5615, 5616, 5617, 5618, 5619, 5620, 5621, 5622, 5623, 5624, 5625, 5626, 5627, 5628, 5629, 5630, 5631, 5632, 5633, 5634, 5635, 5636, 5637, 5638, 5639, 5640, 5641, 5642, 5643, 5644, 5645, 5646, 5647, 5648, 5649, 5650, 5651, 5652, 5653, 5654, 5655, 5656, 5657, 5658, 5659, 5660, 5661, 5662, 5663, 5664, 5665, 5666, 5667, 5668, 5669, 56610, 56611, 56612, 56613, 56614, 56615, 56616, 56617, 56618, 56619, 56620, 56621, 56622, 56623, 56624, 56625, 56626, 56627, 56628, 56629, 56630, 56631, 56632, 56633, 56634, 56635, 56636, 56637, 56638, 56639, 56640, 56641, 56642, 56643, 56644, 56645, 56646, 56647, 56648, 56649, 56650, 56651, 56652, 56653, 56654, 56655, 56656, 56657, 56658, 56659, 56660, 56661, 56662, 56663, 56664, 56665, 56666, 56667, 56668, 56669, 566610, 566611, 566612, 566613, 566614, 566615, 566616, 566617, 566618, 566619, 566620, 566621, 566622, 566623, 566624, 566625, 566626, 566627, 566628, 566629, 566630, 566631, 566632, 566633, 566634, 566635, 566636, 566637, 566638, 566639, 566640, 566641, 566642, 566643, 566644, 566645, 566646, 566647, 566648, 566649, 566650, 566651, 566652, 566653, 566654, 566655, 566656, 566657, 566658, 566659, 566660, 566661, 566662, 566663, 566664, 566665, 566666, 566667, 566668, 566669, 5666610, 5666611, 5666612, 5666613, 5666614, 5666615, 5666616, 5666617, 5666618, 5666619, 5666620, 5666621, 5666622, 5666623, 5666624, 5666625, 5666626, 5666627, 5666628, 5666629, 5666630, 5666631, 5666632, 5666633, 5666634, 5666635, 5666636, 5666637, 5666638, 5666639, 5666640, 5666641, 5666642, 5666643, 5666644, 5666645, 5666646, 5666647, 5666648, 5666649, 5666650, 5666651, 5666652, 5666653, 5666654, 5666655, 5666656, 5666657, 5666658, 5666659, 5666660, 5666661, 5666662, 5666663, 5666664, 5666665, 5666666, 5666667, 5666668, 5666669, 56666610, 56666611, 56666612, 56666613, 56666614, 56666615, 56666616, 56666617, 56666618, 56666619, 56666620, 56666621, 56666622, 56666623, 56666624, 56666625, 56666626, 56666627, 56666628, 56666629, 56666630, 56666631, 56666632, 56666633, 56666634, 56666635, 56666636, 56666637, 56666638, 56666639, 56666640, 56666641, 56666642, 56666643, 56666644, 56666645, 56666646, 56666647, 56666648, 56666649, 56666650, 56666651, 56666652, 56666653, 56666654, 56666655, 56666656, 56666657, 56666658, 56666659, 56666660, 56666661, 56666662, 56666663, 56666664, 56666665, 56666666, 56666667, 56666668, 56666669, 566666610, 566666611, 566666612, 566666613, 566666614, 566666615, 566666616, 566666617, 566666618, 566666619, 566666620, 566666621, 566666622, 566666623, 566666624, 566666625, 566666626, 566666627, 566666628, 566666629, 566666630, 566666631, 566666632, 566666633, 566666634, 566666635, 566666636, 566666637, 566666638, 566666639, 566666640, 566666641, 566666642, 566666643, 566666644, 566666645, 566666646, 566666647, 566666648, 566666649, 566666650, 566666651, 566666652, 566666653, 566666654, 566666655, 566666656, 566666657, 566666658, 566666659, 566666660, 566666661, 566666662, 566666663, 566666664, 566666665, 566666666, 566666667, 566666668, 566666669, 5666666610, 5666666611, 5666666612, 5666666613, 5666666614, 5666666615, 5666666616, 5666666617, 5666666618, 5666666619, 5666666620, 5666666621, 5666666622, 5666666623, 5666666624, 5666666625, 5666666626, 5666666627, 5666666628, 5666666629, 5666666630, 5666666631, 5666666632, 5666666633, 5666666634, 5666666635, 5666666636, 5666666637, 5666666638, 5666666639, 5666666640, 5666666641, 5666666642, 5666666643, 5666666644, 5666666645, 5666666646, 5666666647, 5666666648, 5666666649, 5666666650, 5666666651, 5666666652, 5666666653, 5666666654, 5666666655, 5666666656, 5666666657, 5666666658, 5666666659, 5666666660, 5666666661, 5666666662, 5666666663, 5666666664, 5666666665, 5666666666, 5666666667, 5666666668, 5666666669, 56666666610, 56666666611, 56666666612, 56666666613, 56666666614, 56666666615, 56666666616, 56666666617, 56666666618, 56666666619, 56666666620, 56666666621, 56666666622, 56666666623, 56666666624, 56666666625, 56666666626, 56666666627, 56666666628, 56666666629, 56666666630, 56666666631, 56666666632, 56666666633, 56666666634, 56666666635, 56666666636, 56666666637, 56666666638, 56666666639, 56666666640, 56666666641, 56666666642, 56666666643, 56666666644, 56666666645, 56666666646, 56666666647, 56666666648, 56666666649, 56666666650, 56666666651, 56666666652, 56666666653, 56666666654, 56666666655, 56666666656, 56666666657, 56666666658, 56666666659, 56666666660, 56666666661, 56666666662, 56666666663, 56666666664, 56666666665, 56666666666, 56666666667, 56666666668, 56666666669, 566666666610, 566666666611, 566666666612, 566666666613, 566666666614, 566666666615, 566666666616, 566666666617, 566666666618, 566666666619, 566666666620, 566666666621, 566666666622, 566666666623, 566666666624, 566666666625, 566666666626, 566666666627, 566666666628, 566666666629, 566666666630, 566666666631, 566666666632, 566666666633, 566666666634, 566666666635, 566666666636, 566666666637, 566666666638, 566666666639, 566666666640, 566666666641, 566666666642, 566666666643, 566666666644, 566666666645, 566666666646, 566666666647, 566666666648, 566666666649, 566666666650, 566666666651, 566666666652, 566666666653, 566666666654, 566666666655, 566666666656, 566666666657, 566666666658, 566666666659, 566666666660, 566666666661, 566666666662, 566666666663, 566666666664, 566666666665, 566666666666, 566666666667, 566666666668, 566666666669, 5666666666610, 5666666666611, 5666666666612, 5666666666613, 5666666666614, 5666666666615, 5666666666616, 5666666666617, 5666666666618, 5666666666619, 5666666666620, 5666666666621, 5666666666622, 5666666666623, 5666666666624, 5666666666625, 5666666666626, 5666666666627, 5666666666628, 5666666666629, 5666666666630, 5666666666631, 5666666666632, 5666666666633, 5666666666634, 5666666666635, 5666666666636, 5666666666637, 5666666666638, 5666666666639, 5666666666640, 5666666666641, 5666666666642, 5666666666643, 5666666666644, 5666666666645, 5666666666646, 5666666666647, 5666666666648, 5666666666649, 5666666666650, 5666666666651, 5666666666652, 5666666666653, 5666666666654, 5666666666655, 5666666666656, 5666666666657, 5666666666658, 5666666666659, 5666666666660, 5666666666661, 5666666666662, 5666666666663, 5666666666664, 5666666666665, 5666666666666, 5666666666667, 5666666666668, 5666666666669, 56666666666610, 56666666666611, 56666666666612, 56666666666613, 56666666666614, 56666666666615, 56666666666616, 56666666666617, 56666666666618, 56666666666619, 56666666666620, 56666666666621, 56666666666622, 56666666666623, 56666666666624, 56666666666625, 56666666666626, 56666666666627, 56666666666628, 56666666666629, 56666666666630, 56666666666631, 56666666666632, 56666666666633, 56666666666634, 56666666666635, 56666666666636, 56666666666637, 56666666666638, 56666666666639, 56666666666640, 56666666666641, 56666666666642, 56666666666643, 56666666666644, 56666666666645, 56666666666646, 56666666666647, 56666666666648, 56666666666649, 56666666666650, 56666666666651, 56666666666652, 56666666666653, 56666666666654, 56666666666655, 56666666666656, 56666666666657, 56666666666658, 56666666666659, 56666666666660, 56666666666661, 56666666666662, 56666666666663, 56666666666664, 56666666666665, 56666666666666, 56666666666667, 56666666666668, 56666666666669, 566666666666610, 566666666666611, 566666666666612, 566666666666613, 566666666666614, 566666666666615, 566666666666616, 566666666666617, 566666666666618, 566666666666619, 566666666666620, 566666666666621, 566666666666622, 566666666666623, 566666666666624, 566666666666625, 566666666666626, 566666666666627, 566666666666628, 566666666666629, 566666666666630, 566666666666631, 566666666666632, 566666666666633, 566666666666634, 566666666666635, 566666666666636, 566666666666637, 566666666666638, 566666666666639, 566666666666640, 566666666666641, 566666666666642, 566666666666643, 566666666666644, 566666666666645, 566666666666646, 566666666666647, 566666666666648, 566666666666649, 566666666666650, 566666666666651, 566666666666652, 566666666666653, 566666666

# L'OFFERTA DIDATTICA HOEPLI

L'edizione **Openschool** Hoepli offre a docenti e studenti tutte le potenzialità di Openschool Network (ON), il nuovo sistema integrato di contenuti e servizi per l'apprendimento.

## Edizione **OPENSCHOOL**



### LIBRO DI TESTO



Il libro di testo è l'**elemento cardine** dell'offerta formativa, uno strumento didattico **agile** e **completo**, utilizzabile **autonomamente** o in combinazione con il ricco **corredo digitale** offline e online. Secondo le più recenti indicazioni ministeriali, volume cartaceo e apparati digitali **sono integrati in un unico percorso didattico**. Le espansioni accessibili attraverso l'eBook+ e i materiali integrativi disponibili nel sito dell'editore sono puntualmente richiamati nel testo tramite apposite icone.



### eBOOK+



L'eBook+ è la versione digitale e interattiva del libro di testo, utilizzabile su **tablet**, **LIM** e **computer**. Aiuta a comprendere e ad approfondire i contenuti, rendendo l'apprendimento più attivo e coinvolgente. Consente di leggere, annotare, sottolineare, effettuare ricerche e accedere direttamente alle numerose **risorse digitali integrate**. ➔ Scaricare l'eBook+ è molto **semplice**. È sufficiente seguire le istruzioni riportate nell'ultima pagina di questo volume.



### RISORSE ONLINE



Il sito della casa editrice offre una ricca dotazione di **risorse digitali** per l'approfondimento e l'aggiornamento. Nella pagina web dedicata al testo è disponibile **MyBookBox**, il contenitore virtuale che raccoglie i materiali integrativi che accompagnano l'opera. ➔ Per accedere ai materiali è sufficiente registrarsi al sito **www.hoepliscuola.it** e inserire il codice coupon che si trova nella terza pagina di copertina. **Per il docente** nel sito sono previste ulteriori risorse didattiche dedicate.



### PIATTAFORMA DIDATTICA



La **piattaforma didattica** è un ambiente digitale che può essere utilizzato in modo duttile, a misura delle esigenze della classe e degli studenti. Permette in particolare di **condividere contenuti** ed **esercizi** e di partecipare a **classi virtuali**. Ogni attività svolta viene salvata sul **cloud** e rimane sempre disponibile e aggiornata. La piattaforma consente inoltre di consultare la versione online degli eBook+ presenti nella propria libreria. ➔ È possibile accedere alla piattaforma attraverso il sito **www.hoepliscuola.it**.



# Indice

## UNITÀ DI APPRENDIMENTO 1

### Rappresentazione delle informazioni

#### L1 Comunichiamo con il calcolatore

Introduzione .....	2
La comunicazione .....	3
Tipologia dell'informazione .....	5
Simbologia e terminologia .....	5
Protocollo di comunicazione .....	10
Cenni sulla trasmissione e sul disturbo .....	12
<b>Verifichiamo le conoscenze</b> .....	13
<b>Verifichiamo le competenze</b> .....	14

#### L2 Digitale e binario

Analogico e digitale .....	15
Perché il digitale? .....	18
Digitale o binario? .....	19
Codifica in bit o binaria .....	20
Rappresentazione dei dati alfabetici .....	21
<b>Verifichiamo le conoscenze</b> .....	24
<b>Verifichiamo le competenze</b> .....	25

#### L3 Sistemi di numerazione posizionali

Rappresentazione dei dati numerici .....	26
Sistema additivo/sottrattivo .....	28
Sistema posizionale .....	30
<b>Verifichiamo le conoscenze</b> .....	38
<b>Verifichiamo le competenze</b> .....	39

#### L4 Conversione di base decimale

Introduzione alle conversioni di base .....	40
Conversione in decimale .....	40
Conversione da binario a decimale .....	41
Conversione da ottale a decimale .....	42
Conversione da esadecimale a decimale .....	43

Conversione da decimale intero  
alle diverse basi .....

44

Conversione da decimale a binario .....

44

Conversione da decimale a esadecimale .....

46

Conversione da decimale frazionale  
alle diverse basi .....

47

Conclusioni .....

51

**Verifichiamo le competenze** .....

52

#### L5 Conversione tra le basi binarie

Introduzione .....	54
Conversione tra binari e ottali .....	55
Conversione tra binari ed esadecimali .....	58
Conversione tra ottali ed esadecimali .....	61
<b>Verifichiamo le competenze</b> .....	63

#### L6 Immagini, suoni e filmati

Introduzione .....	64
Immagini digitali .....	65
Compressione delle immagini .....	74
Immagine vettoriale .....	76
Filmati digitali .....	77
Suoni digitali .....	78
Esempio riepilogativo .....	81
<b>Verifichiamo le conoscenze</b> .....	82
<b>Verifichiamo le competenze</b> .....	82

## Esercitazioni di laboratorio

- 1 La compressione dei dati con Huffman .....
- 2 Antichi sistemi di numerazione .....
- 3 I numeri romani con Excel .....
- 4 La conversione di numeri in binario con Excel .....
- 5 Conversione alla base decimale .....
- 6 Conversione tra le basi binarie .....

Puoi scaricare il Lab. 2 anche da



## AREA digitale

-  Esercizi
-  Immagini
-  La compressione dei dati
- ▶ Codifica di Huffman
- ▶ Prefissi binari per il byte
- ▶ Contare fino a 31 con una mano
- ▶ Confronto sulla codifica dei primi 16 numeri
- ▶ Tabella dei colori RGB
- ▶ Esercizi per il recupero
- ▶ Esercizi per l'approfondimento

## UNITÀ DI APPRENDIMENTO 2

### I codici digitali

#### L1 Codici digitali pesati

Introduzione alla codifica dell'informazione .....	108
La codifica di caratteri: codici ASCII e Unicode .....	112
Il codice BCD (Binary Coded Decimal) .....	114
Il codice Aiken .....	117
I codici quinario e biquinario .....	118
Il codice 2 su 5 .....	119
<b>Verifichiamo le conoscenze</b> .....	120
<b>Verifichiamo le competenze</b> .....	121

#### L2 Codici digitali non pesati

Generalità .....	122
Il codice eccesso 3 .....	122
La codifica di Gray .....	124
Il codice eccesso 3 riflesso .....	125
Codice BCD di Petherick .....	126
Codici progressivi: tabella riepilogativa .....	126
Il codice 1 su n .....	127
Il codice a sette segmenti .....	127
Il codice a matrice di punti .....	129
Barcode e QR Code .....	129
HCCB o Microsoft Tag .....	132
<b>Verifichiamo le competenze</b> .....	134

#### L3 La correzione degli errori

Introduzione .....	135
--------------------	-----

Definizioni fondamentali .....	136
Identificazione e correzione degli errori .....	142
<b>Verifichiamo le conoscenze</b> .....	150
<b>Verifichiamo le competenze</b> .....	151

## Esercitazioni di laboratorio

- 1 Simulare una trasmissione seriale con Excel .....
- 2 Calcolo del check digit nei codici EAN 8 ed EAN 13 .....
- 3 Un programma per la codifica di Hamming .....

## AREA digitale

-  Esercizi
-  Tabella caratteri ASCII standard ed esteso
- ▶ Confronto sulla codifica dei numeri da 0 a 9 nei diversi codici
- ▶ Come generare e leggere i QR Code
- ▶ Errori e probabilità
- ▶ Combinazioni a distanza 2

## UNITÀ DI APPRENDIMENTO 3

### La codifica dei numeri

#### L1 Operazioni tra numeri binari senza segno

Aritmetica binaria .....	166
Complemento a 1 .....	166
Complemento a 2 .....	167
Addizione .....	168
Sottrazione .....	169
Prodotto .....	171
Divisione .....	173
<b>Verifichiamo le competenze</b> .....	177

#### L2 Numeri binari relativi

Introduzione .....	178
Modulo e segno .....	178
Complemento alla base .....	182
Eccesso $2^{n-1}$ .....	189
<b>Verifichiamo le competenze</b> .....	192

#### L3 Numeri reali in virgola mobile

I numeri reali in virgola mobile .....	194
--	-----



La codifica binaria dei numeri reali in virgola mobile .....	197
Codifica della mantissa .....	197
Codifica dell'esponente .....	199
Rappresentazione in floating point nello standard IEEE-P754 .....	200
Overflow e underflow .....	204
Conversione da float a decimali .....	205
Errori e arrotondamento .....	209
<b>Verifichiamo le competenze</b> .....	<b>213</b>

## Esercitazioni di laboratorio

1 Facciamo il complemento a uno e a due con Excel .....	217
2 L'addizione e la sottrazione in base binaria .....	221

## AREA *digitale*



- ▶ Circuito sommatore
- ▶ Float nel turbo C++ e Dev-Cpp
- ▶ Intervalli di rappresentazione
- ▶ Esercizi per il recupero
- ▶ Esercizi per l'approfondimento

## UNITÀ DI APPRENDIMENTO 4

### Il sistema operativo

#### L1 Generalità sui sistemi operativi

Accendiamo il PC .....	226
Il sistema operativo .....	228
Kernel .....	231
Shell .....	231
I sistemi operativi in commercio .....	233
<b>Verifichiamo le conoscenze</b> .....	<b>234</b>



#### L2 Evoluzione dei sistemi operativi

- Cenni storici
  - Sistemi dedicati
  - Gestione a lotti (1955-1965)
  - Sistemi interattivi (1965-1980)
  - Home computing (anni Settanta)
  - Sistemi dedicati (anni Ottanta)
  - Sistemi odierni e sviluppi futuri
- Verifichiamo le conoscenze**

#### L3 La gestione del processore

Introduzione al multitasking .....	235
I processi .....	236
Stato dei processi .....	238
La schedulazione dei processi .....	239
User mode e kernel mode .....	241
I criteri di scheduling .....	241
Scheduling a confronto tra sistemi operativi .....	249
Cenni alle problematiche di sincronizzazione .....	249
<b>Verifichiamo le conoscenze</b> .....	<b>251</b>

#### L4 La gestione della memoria

Introduzione .....	252
Caricamento del programma .....	253
Allocazione della memoria: il partizionamento .....	256
Memoria virtuale: introduzione .....	259
Memoria virtuale: paginazione .....	260
Memoria virtuale: segmentazione .....	263
<b>Verifichiamo le conoscenze</b> .....	<b>267</b>

#### L5 La memoria secondaria: il file system

Introduzione .....	268
Il concetto di file .....	269
Struttura della directory .....	272
File nei sistemi multiutente .....	274
Diritti e protezione dei file .....	275
<b>Verifichiamo le conoscenze</b> .....	<b>276</b>



#### L6 Struttura e realizzazione di un file system

- Struttura del file system
  - Il disco fisso
  - Allocazione di un file
  - Realizzazione del file system
- Verifichiamo le conoscenze**



#### L7 La sicurezza del file system

- La sicurezza del file system
  - Struttura di memoria terziaria
- Verifichiamo le conoscenze**



#### L8 La gestione della I/O

- Introduzione
- L'hardware di I/O

Trasferimento dati  
Il sottosistema di I/O del kernel  
**Verifichiamo le conoscenze**

## Esercitazioni di laboratorio

- 1 La shell dei comandi di Windows.....277
- 2 I comandi principali.....285
- 3 I caratteri jolly, reindirizzamento e pipelining.....299
- 4 I file batch.....306

Puoi scaricare le lezioni 2, 6, 7, 8 anche da  hoepiscuola.it

## AREA digitale

-  Esercizi
-  Immagini
-  Process control block
  - Cambio di contesto
  - Esempio di schedulazione di un progetto con Gantt
  - Highest Response Ratio Next Scheduling
  - Periodo dell'RTC
  - Il primo calcolatore con disco magnetico
  - Condivisione di file
  - Accesso indicizzato
  - Flag dei diritti nei file Unix
  - Correggere lo stato dei supporti di memoria
  - Creare un'Unità virtuale
  - Esempio di comandi CD, MD e RD
  - La gestione delle utenze
  - Esempi dei comandi copy, del, ren
  - Caratteri jolly nel comando dir
  - Esempi di pipelining

## UNITÀ DI APPRENDIMENTO 5

### Fasi e modelli di gestione di un ciclo di sviluppo

#### L1 Modelli classici di sviluppo di sistemi informatici

- Introduzione.....316
- Il mestiere del programmatore.....317
- Ingegneria del software e ciclo di vita.....318
- Modello a cascata.....322
- Modello a prototipazione rapida.....323
- Modello incrementale.....324

- Modello a spirale.....325
- Sviluppo “agile” o iterativo incrementale.....326
- Conclusioni.....328
- Verifichiamo le conoscenze**.....329

#### L2 Un nuovo modello di sviluppo: OOP

- Introduzione.....330
- Perché tanti linguaggi di programmazione.....332
- Crisi, dimensioni e qualità del software.....333
- Astrazione, oggetti e classi.....335
- Dov’è la novità?.....336
- Conclusione: che cos’è la programmazione a oggetti.....337
- Verifichiamo le conoscenze**.....340

#### L3 Documentazione di un progetto

- La documentazione del software.....341
- Documentare un progetto a OOP.....342
- Le schede CRC per le classi e le associazioni tra di esse.....343
- Il linguaggio di modellazione UML.....345
- Un esempio completo: aule scolastiche.....353
- Conclusione.....355
- Verifichiamo le conoscenze**.....356
- Verifichiamo le competenze**.....357

## Esercitazioni di laboratorio

- 1 I diagrammi UML con ArgoUML.....358
- 2 Il Model-Oriented Programming con Umple.....366

## AREA digitale

-  Il processo unificato
- Elenco dei documenti di un progetto
- La qualità del software con la norma ISO 9000
- L'UML come metamodello
- Articolo di Timothy Lethbridge su MOP & Umple
- Esercizi per il recupero
- Esercizi per l’approfondimento

**Come utilizzare il coupon per scaricare la versione digitale del libro (eBook+) e i contenuti digitali integrativi (risorse online)**.....372

## 1

# Rappresentazione delle informazioni

- L1 **Comunichiamo con il calcolatore**
- L2 **Digitale e binario**
- L3 **Sistemi di numerazione posizionali**
- L4 **Conversione di base decimale**
- L5 **Conversione tra le basi binarie**
- L6 **Immagini, suoni e filmati**

## Esercitazioni di laboratorio

- ① La compressione dei dati con Huffman; ② Antichi sistemi di numerazione; ③ I numeri romani con Excel; ④ La conversione di numeri in binario con Excel; ⑤ Conversione alla base decimale; ⑥ Conversione tra le basi binarie

### Conoscenze

- Sistema di numerazione decimale, binario, ottale, esadecimale
- Acquisire il concetto di comunicazione
- 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
- Codifica di immagini, suoni e filmati

### Competenze

- Codificare e decodificare numeri e codici
- Codificare i numeri nelle diverse basi
- Convertire numeri e codici rappresentati secondo sistemi diversi
- Convertire un numero in base decimale
- Convertire da binario e ottale in esadecimale
- Distinguere le modalità di codifica dei suoni

### Abilità

- Rappresentare i dati alfabetici
- Effettuare la conversione da basi pesate a decimale
- Effettuare la conversione da decimale a basi pesate di numeri interi e frazionali
- Calcolare l'occupazione di memoria di immagini digitali
- Calcolare l'occupazione di memoria di suoni digitali

## AREA *digitale*



Esercizi



Immagini



- ▶ La compressione dei dati
- ▶ Codifica di Huffman
- ▶ Prefissi binari per il byte
- ▶ Contare fino a 31 con una mano
- ▶ Confronto sulla codifica dei primi 16 numeri
- ▶ Tabella dei colori RGB
- ▶ Esercizi per il recupero
- ▶ Esercizi per l'approfondimento



Soluzioni (prova adesso, esercizi, verifiche)

Puoi scaricare il file anche da [hoepliscuola.it](http://hoepliscuola.it)

# Comunichiamo con il calcolatore

In questa lezione impareremo...

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

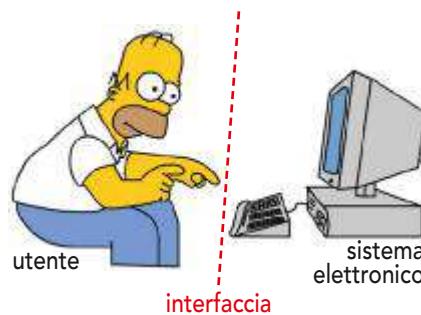
## ■ Introduzione

Il **computer** è una dispositivo elettronico che l'uomo utilizza per le sue due principali caratteristiche:

- **velocità**;
- **capacità di memorizzazione**.

Mediante programmi applicativi, il **software**, l'uomo usa la macchina per fargli eseguire compiti in molteplici aree applicative, dato con oggigiorno pressoché ogni apparecchiatura ha al suo interno una **CPU**, cioè un calcolatore elettronico.

L'utente ha bisogno di comunicare col calcolatore sia per “istruirlo”, scrivendo e memorizzando in esso i programmi, sia per seguire l’evoluzione dei programmi, inserendo dati che devono essere elaborati per produrre i risultati desiderati, che siano programmi applicativi di calcolo, elaborazione testi, di grafica e disegno, di creazione musicale, posta elettronica e così via.



La comunicazione tra uomo e macchina risulta essere ostacolata dalla “differenza” intrinseca dei due sistemi che devono dialogare: se già è problematico comunicare con esseri umani che parlano lingue differenti ancora maggiori saranno le difficoltà che incontriamo per “farcì capire” da un dispositivo elettronico e per comprendere quello che esso vuole comunicare a noi.

È necessario stabilire una **forma di comunicazione** che permetta a un uomo di poter trasmettere un messaggio a un elaboratore automatico ed essere in grado di interpretare le eventuali risposte e un sistema che “si interponga” e permetta a due sistemi diversi di interagire: questo sistema prende il nome di **interfaccia**.



### 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**.

### ESEMPIO

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).

Lo schema è il seguente:



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.

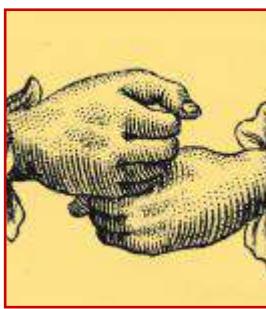
In questa prima parte ci dedicheremo alla **codifica del messaggio**, cioè individueremo che cosa trasmettere o elaborare e lo trasformeremo in formato opportuno per essere elaborato da un **sistema di calcolo**: ci occuperemo di come **codificare l'informazione** a partire dalla definizione dei simboli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

### ESEMPIO

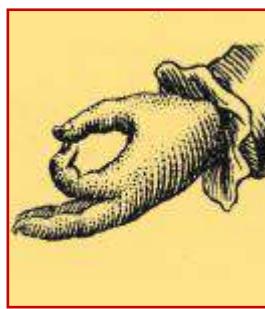
Supponiamo che durante una verifica a scelta multipla vogliate comunicare i risultati a un compagno senza che il docente se ne accorga.

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:



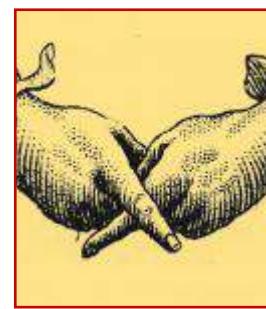
Risposta A



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**.

## ■ 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

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

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

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

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  $|I|$  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  $|I|$  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:

►  $T = (x_1, \dots, x_n)$  alfabeto sorgente, cardinalità  $n$ , cioè composto da  $n$  elementi;

►  $E = (a_1, \dots, a_k)$  alfabeto in codice, cardinalità  $k$ , quindi composto da  $k$  elementi.

La parola codice ha una lunghezza  $l_i = m = \text{costante}$  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:

$$n = \text{INT}_{\max}(\log_k n)$$

Per esempio:

- con due simboli e  $n = 7$  parole è necessario avere  $m = \text{INT}_{\max}[\log_2 7] = 3$ ;
- con tre simboli e  $n = 12$  parole è necessario avere  $m = \text{INT}_{\max}[\log_3 12] = 3$ ;
- 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):

- $T = (\text{Lunedì}, \text{Martedì}, \text{Mercoledì}, \text{Giovedì}, \text{Venerdì}, \text{Sabato}, \text{Domenica})$  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**.



### Prova adesso!

- Codifica a lunghezza fissa

Codifica mesi dell'anno utilizzando un alfabeto di due simboli (A, B):

T = (Gennaio, Febbraio, Marzo ... Dicembre ) cardinalità **n = 12**;

E = (A, B) cardinalità **k = 2**.

Successivamente ripeti la medesima codifica ma utilizzando tre simboli (A,B,C)

E = (A, B, C) cardinalità **k = 3**.

Cosa puoi osservare dalle due diverse codifiche?



### 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:

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

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

$$\begin{array}{lll} x_1 = a & x_4 = b & x_7 = c \\ x_2 = a & x_5 = b & x_8 = c \\ x_3 = a & x_6 = b & x_9 = c \end{array}$$

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:

$$\begin{array}{lll} x_1 = a \ a & x_4 = b \ a & x_7 = c \ a \\ x_2 = a \ b & x_5 = b \ b & x_8 = c \ b \\ x_3 = a \ c & x_6 = b \ c & x_9 = c \ c \end{array}$$

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

### AREA digitale



La compressione dei dati

### AREA digitale



Codifica di Huffman

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.



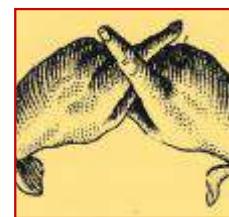
inizio comunicazione



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.

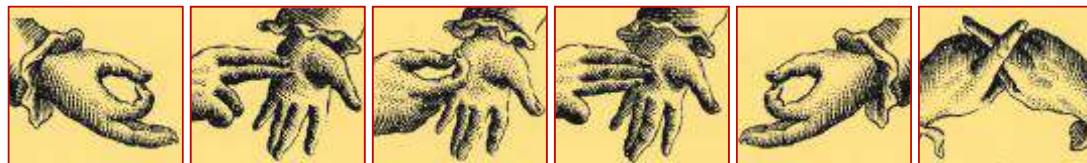


domanda 2



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

### 1. Risposta multipla

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

- a. messaggio
- b. trasmettitore
- c. ricevitore
- d. canale
- e. codice
- f. alfabeto
- g. protocollo

**2** Con il termine "codifica" si intende:

- a. una qualunque sequenza di simboli dell'alfabeto in codice
- b. una qualunque sequenza di simboli dell'alfabeto sorgente
- c. una qualunque sequenza di simboli da trasmettere
- d. nessuna delle affermazioni precedenti

**3** Nella codifica a lunghezza fissa:

- a. le parole dell'alfabeto sorgente hanno lunghezza costante
- b. le parole dell'alfabeto in codice hanno lunghezza costante
- c. i caratteri dell'alfabeto sorgente hanno lunghezza costante
- d. i caratteri dell'alfabeto in codice hanno lunghezza costante

**4** Quali affermazioni sul codice di Huffman sono false?

- a. genera parole in codice di lunghezza variabile
- b. utilizza un alfabeto in codice di lunghezza variabile
- c. viene utilizzato nella compressione dei dati
- d. è un codice ridondante
- e. le parole dell'alfabeto sorgente possono avere frequenze diverse

**5** Quale componente non appartiene al protocollo di comunicazione?

- a. segnale di inizio comunicazione
- b. segnale di fine comunicazione
- c. segnale di sincronismo
- d. segnale di errore nella trasmissione
- e. strutturazione del messaggio

### 2. Vero o 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

### 1. Esercizi

- 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}.
- 12** Individua le frequenze di ogni carattere nella seguente frase da trasmettere:  
"Ambarabaciccicoccò, tre civette sul comò".  
Individua una codifica a lunghezza variabile che permetta di risparmiare almeno il 25% di spazio rispetto a una codifica a lunghezza fissa con un alfabeto di soli due simboli.

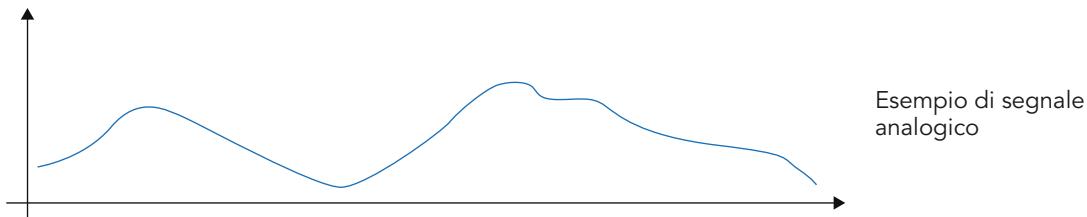
# Digitale e binario

In questa lezione 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.



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

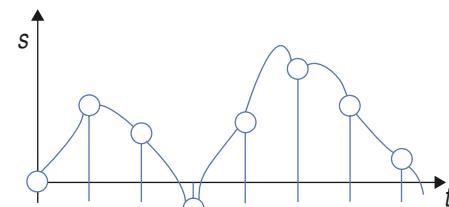
Grandezza	Segnale analogico
Tempo (secondi)	Angolo generato dalla lancetta dell'orologio
Musica	Segnale elettrico microfonico
Luce	Apertura dell'otturatore, apertura della pupilla
Temperatura	Altezza colonna di mercurio del termometro
Velocità	Lancetta del contachilometri

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.

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 “rilevi” 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



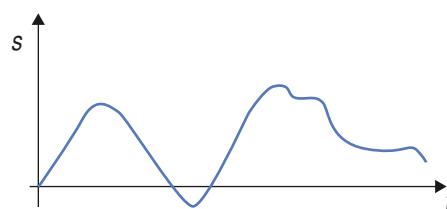
### SEGNALI

**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.

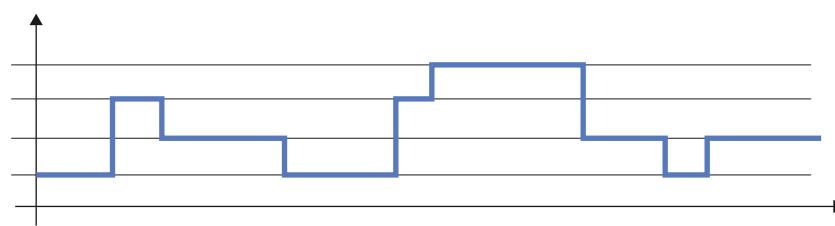


Segnale analogico tempo-continuo



Segnale digitale tempo-discreto

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.



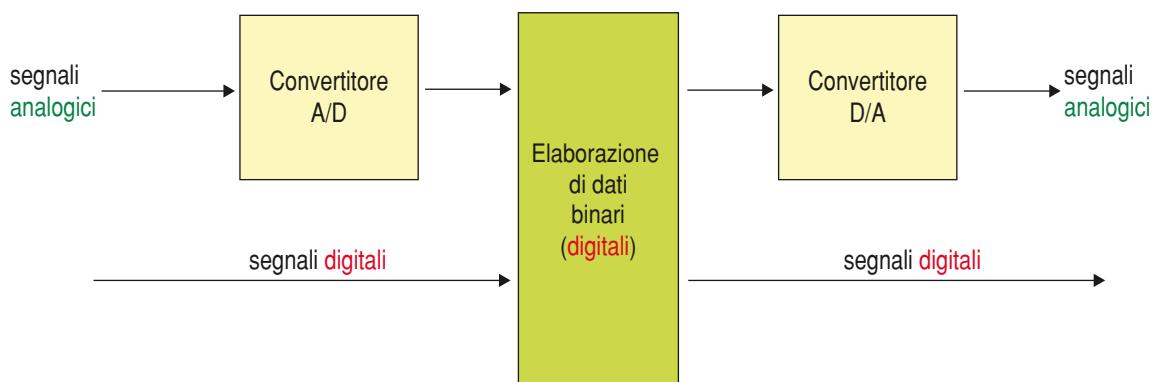
### DIGITALE

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:

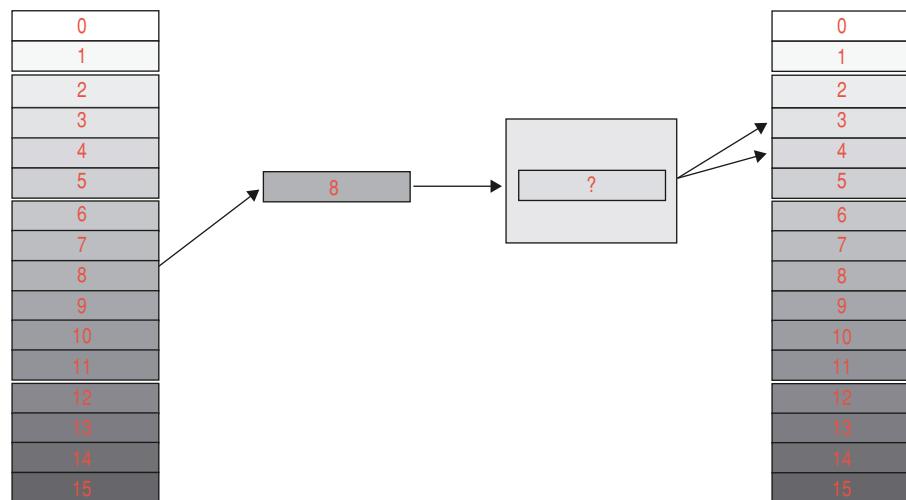
- è possibile elaborare i dati con il computer, in quanto essi sono rappresentati in formato numerico;
- è possibile integrare diverse fonti, quali musica, parole, immagini, in un unico supporto come il CD-ROM;
- si possono condividere le informazioni attraverso la rete Internet.

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.

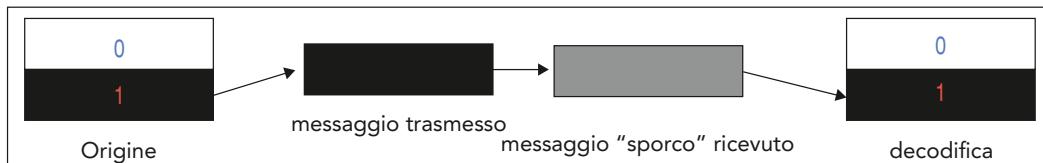
### ESEMPIO

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 8 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.

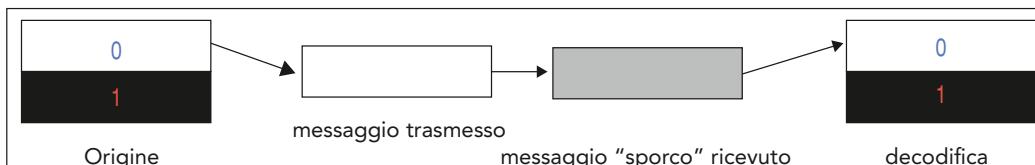


Lo stesso problema può essere risolto con un sistema digitale, dove i segnali hanno per esempio solo due possibili valori (**binari**). Se viene trasmesso per esempio il nero e per un qualunque motivo il segnale si "sporca" prima di essere ricevuto, abbiamo un margine

molto 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.



### CODIFICA

Con il termine **codifica** intendiamo il processo che assegna un codice univoco a tutti gli oggetti di un insieme predefinito utilizzando un insieme di simboli chiamato “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  $\rightarrow 2 = 2^1$  informazioni (0, 1);
- utilizzando una sequenza di 2 bit possiamo avere quattro codifiche distinte:
  - 2 bit  $\rightarrow 4 = 2^2$  informazioni, ottenuti dalle combinazioni di più 0 e 1 (00, 01, 10, 11);
- utilizzando 3 bit possiamo rappresentare otto informazioni differenti:
  - 3 bit  $\rightarrow 8 = 2^3$  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 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.

### 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.

Quindi:

- ▶ il 7 di quadri viene codificato con **011101**;
- ▶ il 4 di picche viene codificato con **010011**;
- ▶ ecc.

♥	00
♦	01
♣	10
♠	11



## Prova adesso!

- Codici binari

Prova a codificare in formato binario:

- 1 i colori dell'arcobaleno;
- 2 i mesi dell'anno;
- 3 le pedine della dama.

## ■ 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 ( $2^7 = 128$ ).

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

### ESEMPIO

Tra i caratteri di controllo ne ricordiamo due “per motivi storici” che permettono di gestire la carta nelle stampanti a impatto:

- codice ASCII 12 = avanzamento di pagina (FF): usato per fare il salto di una pagina;
- codice ASCII 13 = ritorno a capo (CR): usato per far avanzare la scrittura nelle stampanti a impatto.

Tra la codifica delle lettere maiuscole e quella delle lettere minuscole abbiamo “il 6° bit” di differenza, dato che la loro distanza è 32 in base decimale. Se:

- 6° bit = 0 → lettera maiuscola
- 6° bit = 1 → lettera minuscola

Byte	Cod	Char	Byte	Cod	Char	Byte	Cod	Char	Byte	Cod	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
00011101	29	Group separator	00111101	61	=	01011101	93	J	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	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

0110011101101001011000010110111100101110

viene scomposta in byte

0110011101101001011000010110111100101110

e decodificata in

c i a o .

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



## Prova adesso!

• Codice ASCII

- 1 Prova a codificare in formato ASCII:
  - a. "Topolino"
  - b. il tuo nome e cognome;
  - c. "Tecnologie Informatiche".
- 2 Prova a decodificare le seguenti parole dal formato ASCII:
  - a. 01010000 01100001 01110000 01100101 01110010 01101001 01101110 01101111
  - b. 01000011 01100001 01101101 01101101 01100101 01101100 01101100 01101111
  - c. 01010000 01101001 01101110 01101011 00100000 01000110 01101100 01101111  
01111001 01100100

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

## EBCDIC

Il codice **EBCDIC** (**E**xtended **B**inary-Coded **D**ecimal **I**nterchange **C**ode) 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 meno che meno quelli dell'arabo, del bengalese, dell'ebraico, del thailandese ecc.

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**.

**AREA** digitale



Prefissi binari per il byte

## Verifichiamo le conoscenze



### 1. Risposta multipla

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

- a. musica
- b. luce
- c. secondi
- d. temperatura
- e. velocità

**2** Un segnale analogico:

- a. è continuo nel tempo
- b. assume solo due valori
- c. assume solo un insieme finito di valori
- d. si codifica con un campionatore

**3** Un segnale digitale:

- a. è continuo nel tempo
- b. assume solo due valori
- c. assume solo un insieme finito di valori
- d. si rappresenta con le dita

**4** Indica quale non è un dato multimediale:

- a. immagine
- b. testo
- c. suono
- d. filmato

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

- a. Automatic Standard Code for Information Interchange
- b. Australian Standard Code for Informatic Interchange
- c. Automatic Standard Code for Informatic Interchange
- d. American Standard Code for Information Interchange

**6** Il codice ASCII a 8 bit:

- a. ha il MSB con valore 1
- b. ha il MSB con valore 0
- c. ha il LSB con valore 1
- d. ha il LSB con valore 0



### 2. Completamento

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

- a. KiB = \_\_\_\_\_ B
- b. KB = \_\_\_\_\_ B
- c. MiB = \_\_\_\_\_ KB
- d. MB = \_\_\_\_\_ KB
- e. GiB = \_\_\_\_\_ MB
- f. GB = \_\_\_\_\_ MB

**2** Esegui le seguenti equivalenze:

- a. 1240 byte = \_\_\_\_\_ K
- b. 1240 KB = \_\_\_\_\_ MB
- c. 4320 MB = \_\_\_\_\_ GB
- d. 4320 MB = \_\_\_\_\_ TB

**3** Esegui le seguenti equivalenze:

- a. 12 KB = \_\_\_\_\_ byte
- b. 24 MB = \_\_\_\_\_ KB
- c. 43 MB = \_\_\_\_\_ byte
- d. 13 TB = \_\_\_\_\_ GB



### 3. Vero o 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

### 1. Esercizi

- 1 Scrivi il procedimento da utilizzare per capire quanti bit sono necessari per rappresentare un valore tra un numero qualunque di alternative (per es. il numero di una pagina tra quelle di un libro).
- 2 Quanti bit sono necessari per individuare un minuto in un anno?
- 3 Quanti bit sono necessari per codificare i giorni della settimana?  
E i giorni del mese?  
E i giorni di un anno?
- 4 Codifica in binario i sette nani e Biancaneve con due diverse configurazioni di bit.
- 5 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).
- 6 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.
- 7 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.
- 8 Codifica in binario utilizzando la tabella dei codici ASCII le seguenti parole:  
Mario  
Cammello  
Zio Pino  
Abracadabra
- 9 Decodifica utilizzando la tabella dei codici ASCII le seguenti stringhe binarie:  
0110110101100001011011010110110101100001  
010100000110000101101110110110001101111  
010000010111010101100111011101010111001001101001  
010001110110110001101111011100100110100101100001
- 10 Codificare gli otto colori seguenti in codice binario: nero, rosso, blu, giallo, verde, viola, grigio, arancione
- 11 Hai ricevuto un messaggio di posta elettronica da un amico. Il messaggio contiene: un testo di 300 caratteri scritto in ASCII; un'immagine di 120 × 150 pixel con 1024 colori. Quanti byte occupa il messaggio?
- 12 Un'immagine di 300 × 400 pixel occupa 15000 byte. L'immagine è a colori oppure in bianco e nero?
- 13 Scrivi le domande binarie (vero/falso, sì/no) minime per saper qual è il giorno della settimana (lunedì, martedì, mercoledì... domenica).
- 14 Scrivi la parola "computer" con una sequenza di codici ASCII.



# Sistemi di numerazione posizionali

In questa lezione impareremo...

- l'origine dei sistemi di numerazione posizionale
- 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  $\Rightarrow$  **significato**;
- **modalità espressiva**: la sua codifica  $\Rightarrow$  **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.

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:

$$\text{MCDXCII} = 1000 - 100 + 500 - 10 + 100 + 1 + 1 = 1492$$

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:

$$\text{XIII} \quad = 10 + 1 + 1 + 1$$

$$\text{V} \quad = 5$$

$$\text{MCMLXXXI} \quad = 1000 - 100 + 1000 + 50 + 10 + 10 + 10 + 1$$



**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 VIIIIL 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.

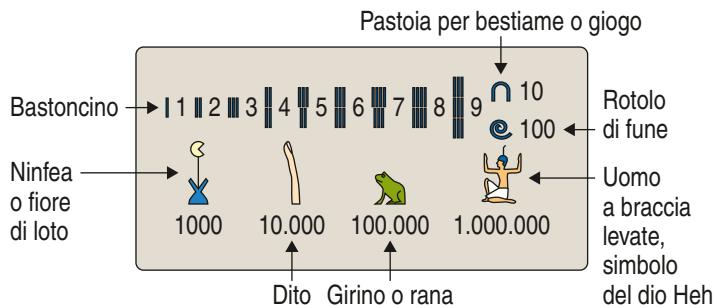


## Prova adesso!

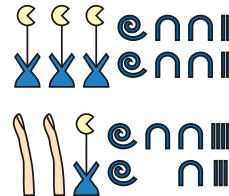
- Conversione da decimale a romano
- Conversione da romano a decimale

- 1 Esegui la conversione nel sistema numerale romano dei seguenti numeri:  
24, 12, 88, 44, 69, 432, 1548
- 2 Esegui la conversione del sistema numerale romano dei seguenti numeri:  
LXII, DCLXVI, LVI, CMLXVII, DCCLXXVI, LXXXV, CCXLIII

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 seguente:



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**.



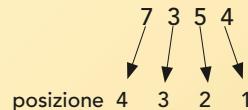
### SISTEMA POSIZIONALE

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*).

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.

La **posizione** di ogni singola cifra si conta da destra verso sinistra, a partire dalla posizione 1 (cifra meno significativa), come indicato nel seguente esempio:



Per poter correttamente interpretare una sequenza di cifre è necessario indicare il **pedice** che indica la sua base: senza **pedice** rappresenta un numero naturale espresso in base 10.

Le basi utilizzate in informatica sono:

- **binaria** (o base 2): utilizza due simboli (0,1) e un numero binario viene indicato con  $1011_2$ ;
- **ottale** (o base 8): utilizza otto simboli (0,1, 2, 3, 4, 5, 6, 7) e un numero binario viene indicato con  $1057_8$ ;
- **esadecimale** (o base 16): utilizza 16 simboli (0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) e un numero binario viene indicato  $4E_{16}$ .

Ciascuna cifra **c** presente nel numero deve essere moltiplicata per un numero ottenuto come potenza della base elevato a un **esponente** che tiene conto della posizione che questa

assume nella cifra. Questo numero si chiama **ordine di una cifra** e assume valore  $m = p - 1$ , dove  $p$  è la posizione che la cifra assume nel numero:  $c_0$  è la cifra più a destra, la **meno significativa**, mentre  $c_{m-1}$  è quella più a sinistra, la più significativa. Nell'esempio precedente 7 è cifra **più significativa** in posizione 4, quindi **peso con esponente 3** e 4 come cifra **meno significativa** che è sempre in posizione 1, quindi **peso con esponente 0**.

### ESEMPIO

Il seguente numero in **base 10**, quindi decimale, di quattro cifre  $N_{10} = 7354_{10}$  può essere visto come la somma di quattro addendi, dove ogni cifra viene moltiplicata per il peso associato alla sua posizione.

$$7354_{10} = 7 \cdot 1000 + 3 \cdot 100 + 5 \cdot 10 + 4 \cdot 1 = 7 \cdot 10^3 + 3 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$$

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

Possiamo vedere ogni singolo termine del dettaglio:

<b>moltiplicatore</b>	7	3	5	4
<b>posizione</b>	4	3	2	1
<b>potenze/peso</b>	$10^3$	$10^2$	$10^1$	$10^0$
	1000	100	10	1



### Prova adesso!

Individua tutti gli addendi dei seguenti numeri decimali:

- 1 375
- 2 30.463
- 3 801.912

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 diminuita di 1** della cifra nell'intera rappresentazione numerica: si ottiene quindi 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).

## 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  $\text{IIIIIIIIII} = 12$ .

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:

$\text{IIII} = 4$  può essere visto come

$$1111_1 = 1 \cdot 1^3 + 1 \cdot 1^2 + 1 \cdot 1^1 + 1 \cdot 1^0 = 1 + 1 + 1 + 1 = 4$$

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.

<b>moltiplicatore</b>	1	1	1	1
posizione	4	3	2	1
<b>potenze/peso</b>	<b>1<sup>3</sup></b>	<b>1<sup>2</sup></b>	<b>1<sup>1</sup></b>	<b>1<sup>0</sup></b>
	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>

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 lezioni.

Effettuiamo ora una conversione da numero binario a numero decimale applicando direttamente la definizione di numerazione posizionale. Vediamo solo un esempio di codifica: il numero 1001.

## ESEMPIO

Individuiamo il valore decimale rappresentato del seguente numero binario **1001<sub>2</sub>**:

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

quindi il risultato decimale del numero binario è 9.

Possiamo vedere ogni singolo addendo nel dettaglio:



## Prova adesso!

- Da binario a decimale

Converti in decimale i seguenti numeri espressi in binario:

$$1 \quad 1111_2 = \underline{\hspace{2cm}}_{10}$$

$$2 \quad 1001\ 1010_2 = \underline{\hspace{2cm}}_{10}$$

$$3 \quad 1111\ 0011_2 = \underline{\hspace{2cm}}_{10}$$

### AREA digitale

 Contare fino a 31 con una mano

## 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:

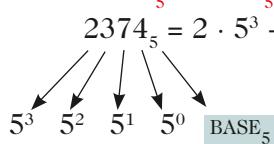
1 = tai	6 = otai	= nuovo uno
2 = lua	7 = olua	= nuovo due
3 = tolu	8 = otolu	= nuovo tre
4 = vari	9 = ovari	= nuovo quattro
5 = luna	10 = lualuna	= due mani ( $2 \times 5$ )

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:

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


 BASE<sub>10</sub>

e come risultato decimale dà il numero decimale 364.

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	2	3	7	4	
posizione	4	3	2	1	
potenze/peso	$5^3$	$5^2$	$5^1$	$5^0$	
	<b>125</b>	<b>25</b>	<b>5</b>	<b>1</b>	
addendi	<b>250</b>	<b>75</b>	<b>35</b>	<b>4</b>	<b>364</b>

## 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.

### ESEMPIO

La stringa  $N_8 = 354_8$  si scrive come somma di tre termini:

$$354_8 = 3 \cdot 8^2 + 5 \cdot 8^1 + 4 \cdot 8^0 = 3 \cdot 64 + 40 + 4 = 192 + 40 + 4 = 236_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

<b>moltiplicatore</b>	0	3	5	4	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$8^3$	$8^2$	$8^1$	$8^0$	
	<b>512</b>	<b>64</b>	<b>8</b>	<b>1</b>	
<b>addendi</b>	<b>0</b>	<b>192</b>	<b>40</b>	<b>4</b>	<b>236</b>

La stringa  $N_8 = 5712_8$  si scrive come 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:

<b>moltiplicatore</b>	5	7	1	2	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$8^3$	$8^2$	$8^1$	$8^0$	
	<b>512</b>	<b>64</b>	<b>8</b>	<b>1</b>	
<b>addendi</b>	<b>2560</b>	<b>448</b>	<b>8</b>	<b>2</b>	<b>3018</b>



### Prova adesso!

• Da ottale a decimale

Converti in decimale i seguenti numeri espressi in ottale:

1  $333_8 = \underline{\hspace{2cm}}_{10}$

2  $777_8 = \underline{\hspace{2cm}}_{10}$

3  $4567_8 = \underline{\hspace{2cm}}_{10}$

## 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.

1	2	3	4	5	6	7	8	9	0
١	٢	٣	٤	٥	٦	٧	٨	٩	٠

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.

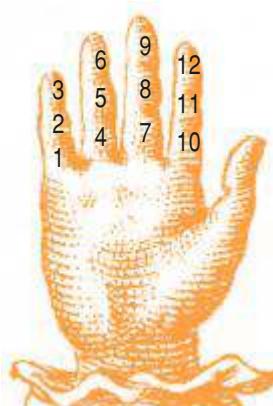
## Il sistema a base 12

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 ( $3 \times 4 = 12$ ).

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  $30^\circ$  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:  $\Sigma = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ .

### ESEMPIO

La stringa  $N_{16} = 3B2_{16}$  si scrive come somma di tre termini:

$$3B2_{16} = 3 \cdot 16^2 + B \cdot 16^1 + 2 \cdot 16^0 = 3 \cdot 256 + 11 \cdot 16 + 2 = 768 + 176 + 2 = 946_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore		3	B	2	
posizione	4	3	2	1	
potenze/peso	$16^3$	$16^2$	$16^1$	$16^0$	
	<b>4096</b>	<b>256</b>	<b>16</b>	<b>1</b>	
addendi		<b>768</b>	<b>176</b>	<b>2</b>	<b>946</b>

La stringa  $N_{16} = 13FA_{16}$  si scrive come 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	A	
posizione	4	3	2	1	
potenze/peso	$16^3$	$16^2$	$16^1$	$16^0$	
	<b>4096</b>	<b>256</b>	<b>16</b>	<b>1</b>	
addendi	<b>4096</b>	<b>768</b>	<b>240</b>	<b>10</b>	<b>5114</b>



### Prova adesso!

• Da esadecimale a decimale

Converti in decimale i seguenti numeri espressi in ottale:

1     $234_H = \underline{\hspace{2cm}}_{10}$

2     $ABC_H = \underline{\hspace{2cm}}_{10}$

3     $F16A_H = \underline{\hspace{2cm}}_{10}$

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  $360^\circ$ ).

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.

## ESEMPIO

Rappresentiamo il numero 100 nelle diverse basi:

Binario:  $100_{10} = 1100100_2$

Ottale:  $100_{10} = 144_8$

Esadecimale:  $100_{10} = 64_H$

## AREA digitale



Confronto sulla codifica dei primi 16 numeri

## Verifichiamo le conoscenze



### 1. Risposta multipla

- 1** Quanti bit occorrono per codificare 18 valori diversi?  
a. 3      b. 4      c. 5      d. 6
- 2** Quanti bit occorrono per codificare i numeri pari da 2 a 16?  
a. 3      b. 4      c. 5      d. 6
- 3** Quanti byte occorrono per codificare 300 informazioni diverse?  
a. 1      b. 2      c. 3      d. 4
- 4** Quante cifre occorrono nel sistema di numerazione a base 7?  
a. 3      b. 6      c. 7      d. 8
- 5** Quale non può essere la base del numero 5732?  
a. 7      b. 8      c. 9      d. 16
- 6** Qual è la base del numero 134201?  
a. 2      b. 3      c. 4      d. 5
- 7** Qual è il valore nel sistema decimale del numero che, in base 4, si scrive 1032<sub>4</sub>?  
a. 34      b. 78      c. 84      d. 98
- 8** Qual è il valore numerico associato alla cifra 1 del numero 20135<sub>8</sub>?  
a. 1      b. 8      c. 64      d. 256



### 2. Vero o falso

- 1** I numeri sono entità matematiche astratte. 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
- 9** Con un byte per la codifica si possono rappresentare 255 informazioni distinte. V F
- 10** La frase "dipartimento di psicologia" scritta in ASCII occupa 26 byte. V F

## Verifichiamo le competenze

### 1. Esercizi

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

$$1101_2 = \underline{\hspace{2cm}}_{10}$$

$$1011_2 = \underline{\hspace{2cm}}_{10}$$

$$11001_2 = \underline{\hspace{2cm}}_{10}$$

$$10101_2 = \underline{\hspace{2cm}}_{10}$$

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

$$1010_2 = \underline{\hspace{2cm}}_{10}$$

$$1111_2 = \underline{\hspace{2cm}}_{10}$$

$$11101_2 = \underline{\hspace{2cm}}_{10}$$

$$11011_2 = \underline{\hspace{2cm}}_{10}$$

$$100001_2 = \underline{\hspace{2cm}}_{10}$$

$$1101011_2 = \underline{\hspace{2cm}}_{10}$$

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

$$01101111_2 = \underline{\hspace{2cm}}_{10}$$

$$11000011_2 = \underline{\hspace{2cm}}_{10}$$

$$11101110_2 = \underline{\hspace{2cm}}_{10}$$

$$11110001_2 = \underline{\hspace{2cm}}_{10}$$

$$10101010_2 = \underline{\hspace{2cm}}_{10}$$

$$11000011_2 = \underline{\hspace{2cm}}_{10}$$

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

$$11_H = \underline{\hspace{2cm}}_{10}$$

$$22_H = \underline{\hspace{2cm}}_{10}$$

$$33_H = \underline{\hspace{2cm}}_{10}$$

$$44_H = \underline{\hspace{2cm}}_{10}$$

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

$$AB_H = \underline{\hspace{2cm}}_{10}$$

$$FF_H = \underline{\hspace{2cm}}_{10}$$

$$12D_H = \underline{\hspace{2cm}}_{10}$$

$$31A1_H = \underline{\hspace{2cm}}_{10}$$

$$1234_H = \underline{\hspace{2cm}}_{10}$$

$$2288_H = \underline{\hspace{2cm}}_{10}$$

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

$$77_8 = \underline{\hspace{2cm}}_{10}$$

$$456_8 = \underline{\hspace{2cm}}_{10}$$

$$2123_8 = \underline{\hspace{2cm}}_{10}$$

$$4567_8 = \underline{\hspace{2cm}}_{10}$$

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

$$22_8 = \underline{\hspace{2cm}}_{10}$$

$$123_8 = \underline{\hspace{2cm}}_{10}$$

$$2143_8 = \underline{\hspace{2cm}}_{10}$$

$$11111_8 = \underline{\hspace{2cm}}_{10}$$

$$12345_8 = \underline{\hspace{2cm}}_{10}$$

$$23456_8 = \underline{\hspace{2cm}}_{10}$$

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

$$44_8 = \underline{\hspace{2cm}}_{10}$$

$$246_8 = \underline{\hspace{2cm}}_{10}$$

$$1357_8 = \underline{\hspace{2cm}}_{10}$$

$$20402_8 = \underline{\hspace{2cm}}_{10}$$

$$23755_8 = \underline{\hspace{2cm}}_{10}$$

$$46325_8 = \underline{\hspace{2cm}}_{10}$$

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

$$4A_H = \underline{\hspace{2cm}}_{10}$$

$$5B_H = \underline{\hspace{2cm}}_{10}$$

$$6C_H = \underline{\hspace{2cm}}_{10}$$

$$7E_H = \underline{\hspace{2cm}}_{10}$$

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

$$BA_H = \underline{\hspace{2cm}}_{10}$$

$$EE_H = \underline{\hspace{2cm}}_{10}$$

$$123_H = \underline{\hspace{2cm}}_{10}$$

$$ABC_H = \underline{\hspace{2cm}}_{10}$$

$$2525_H = \underline{\hspace{2cm}}_{10}$$

$$3AFC_H = \underline{\hspace{2cm}}_{10}$$



# Conversione di base decimale

In questa lezione 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.

## ■ 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

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

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

$$N_{10} = c_{m-1} \cdot b^{m-1} + c_{m-2} \cdot b^{m-2} \dots c_2 \cdot b^2 + c_1 \cdot b^1 + c_0 \cdot b^0$$

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

Abbiamo già parlato del sistema binario dato che è il sistema che utilizziamo per rappresentare i dati nei sistemi digitali: effettuiamo ora una conversione da **numero binario** a **numero decimale** applicando direttamente la definizione di numerazione posizionale.

### ESEMPIO

Individuiamo il valore decimale rappresentato del seguente numero binario  $1101_2$ :

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 13_{10}$$

quindi il risultato decimale del numero binario è 13.

Possiamo vedere ogni singolo addendo nel dettaglio:

moltiplicatore					1	1	0	1	
posizione	8	7	6	5	4	3	2	1	
potenze/peso	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	
addendi	128	64	32	16	8	4	2	1	13

### ESEMPIO

Convertiamo il numero 10.101, scrivendolo sotto forma di somma di termini:  
Possiamo vedere ogni singolo termine nel dettaglio:

$$10.101_2 = 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 2 + 0 + 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} = 2.625$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	0	1	0	1
posizione	2	1	-1	-2	-3
potenze/peso	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$
addendi	2	0	0.5	0	0.125



## Prova adesso!

• Da binario a decimale

Converti in decimale i seguenti numeri espressi in binario:

$$\text{1} \quad 1111_2 = \underline{\hspace{2cm}}_{10}$$

$$\text{2} \quad 1001\ 1010_2 = \underline{\hspace{2cm}}_{10}$$

$$\text{3} \quad 1111\ 0011_2 = \underline{\hspace{2cm}}_{10}$$

$$\text{4} \quad 10.11 = \underline{\hspace{2cm}}_{10}$$

$$\text{5} \quad 101.001 = \underline{\hspace{2cm}}_{10}$$

$$\text{6} \quad 1011.011 = \underline{\hspace{2cm}}_{10}$$

## ■ Conversione da ottale a decimale

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.

### ESEMPIO

La stringa  $N_8 = 354_8$  si scrive come somma di tre termini:

$$354_8 = 3 \cdot 8^2 + 5 \cdot 8^1 + 4 \cdot 8^0 = 3 \cdot 64 + 40 + 4 = 192 + 40 + 4 = 236_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

<b>moltiplicatore</b>	0	3	5	4	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$8^3$	$8^2$	$8^1$	$8^0$	
	<b>512</b>	<b>64</b>	<b>8</b>	<b>1</b>	
<b>addendi</b>	<b>0</b>	<b>192</b>	<b>40</b>	<b>4</b>	<b>236</b>

La stringa  $N_8 = 5712_8$  si scrive come 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:

<b>moltiplicatore</b>	5	7	1	2	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$8^3$	$8^2$	$8^1$	$8^0$	
	<b>512</b>	<b>64</b>	<b>8</b>	<b>1</b>	
<b>addendi</b>	<b>2560</b>	<b>448</b>	<b>8</b>	<b>2</b>	<b>3018</b>

La stringa  $N_8 = 1531.24$  si scrive come somma di sei termini:

$$1531.24_8 = 1 \cdot 8^3 + 5 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0 + 2 \cdot 8^{-1} + 4 \cdot 8^{-2} = 512 + 320 + 24 + 1 + 2 \cdot \frac{1}{8} + 4 \cdot \frac{1}{64} = 857.313_{10}$$

<b>moltiplicatore</b>	1	5	3	1	2	4	
<b>posizione</b>	4	3	2	1	-1	-2	
<b>potenze/peso</b>	$8^3$	$8^2$	$8^1$	$8^0$	$8^{-1}$	$8^{-2}$	
	<b>512</b>	<b>64</b>	<b>8</b>	<b>1</b>	<b>1/8=0.125</b>	<b>1/64=0.0157</b>	
<b>addendi</b>	<b>512</b>	<b>320</b>	<b>24</b>	<b>1</b>	<b>0.25</b>	<b>0.0628</b>	<b>857.313</b>



## Prova adesso!

• Da ottale a decimale

Converti in decimale i seguenti numeri espressi in ottale:

1  $333_8 = \underline{\hspace{2cm}}_{10}$

2  $777_8 = \underline{\hspace{2cm}}_{10}$

3  $4567_8 = \underline{\hspace{2cm}}_{10}$

4  $2.23 = \underline{\hspace{2cm}}_{10}$

5  $35.641 = \underline{\hspace{2cm}}_{10}$

6  $621.326 = \underline{\hspace{2cm}}_{10}$

## ■ Conversione da esadecimale a decimale

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:  $\Sigma = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ .

### ESEMPIO

La stringa  $N_{16} = 3B2_{16}$  si scrive come somma di tre termini:

$$3B2_{16} = 3 \cdot 16^2 + B \cdot 16^1 + 2 \cdot 16^0 = 3 \cdot 256 + 11 \cdot 16 + 2 = 768 + 176 + 2 = 946_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

<b>moltiplicatore</b>		3	B	2	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$16^3$	$16^2$	$16^1$	$16^0$	
	<b>4096</b>	<b>256</b>	<b>16</b>	<b>1</b>	
<b>addendi</b>		<b>768</b>	<b>176</b>	<b>2</b>	<b>946</b>

La stringa  $N_{16} = 13FA_{16}$  si scrive come 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:

<b>moltiplicatore</b>	1	3	F	A	
<b>posizione</b>	4	3	2	1	
<b>potenze/peso</b>	$16^3$	$16^2$	$16^1$	$16^0$	
	<b>4096</b>	<b>256</b>	<b>16</b>	<b>1</b>	
<b>addendi</b>	<b>4096</b>	<b>768</b>	<b>240</b>	<b>10</b>	<b>5114</b>

La stringa  $N_{16} = ABC.25_{16}$  si scrive come somma di cinque termini:

$$ABC.25_{16} = 10 \cdot 16^2 + 11 \cdot 16^1 + 12 \cdot 16^0 + 2 \cdot 16^{-1} + 5 \cdot 16^{-2} =$$

$$= 10 \cdot 256 + 11 \cdot 16 + 12 + 2 \cdot \frac{1}{16} + 5 \cdot \frac{1}{256} = 2748.145_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	0	A=10	B=11	C=12	2	5
posizione	4	3	2	1	-1	-2
potenze/peso	$16^3$	$16^2$	$16^1$	$16^0$	$16^{-1}$	$16^{-2}$
	<b>4096</b>	<b>256</b>	<b>16</b>	<b>1</b>	<b><math>1/16=0.0625</math></b>	<b><math>1/256=0.004</math></b>
addendi	<b>0</b>	<b>2560</b>	<b>176</b>	<b>12</b>	<b>0.125</b>	<b>0.020</b>

## ■ 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.

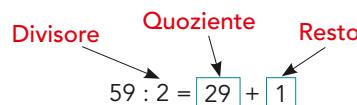
B) Si leggono i resti nell'ordine opposto a quello con cui sono stati trovati.

## ■ Conversione da decimale a binario

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

Il valore di N è 59 quindi non è uguale a 1, perciò:

- ➊ dividiamo il numero 59 per 2, ottenendo il quoziente (29) e il resto (1)



e mettiamo i valori ottenuti in una tabella:

passo	Valore	Divisore	Quoziente	Resto
1	59	2	29	1

2 sostituiamo al valore (59) il quoziente (29) e ripetiamo la divisione per 2:

$$\begin{array}{ccc} \text{Divisore} & \text{Quoziente} & \text{Resto} \\ \searrow & \downarrow & \swarrow \\ 29 : 2 = & 14 & + 1 \end{array}$$

e aggiungiamo una riga nella tabella:

passo	Valore	Divisore	Quoziente	Resto
1	59	2	29	1
2	29	2	14	1

Continuiamo a ripetere questo procedimento fino a che il quoziente diviene 0: completiamo quindi la tabella come quella riportata nella figura seguente:

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 2, cioè la lettura dei resti in ordine opposto, dall'ultimo al primo (dal basso verso l'alto):

$$N_2 = 1\ 1\ 1\ 0\ 1\ 1$$

Quindi abbiamo completato la conversione ottenendo:

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

Nel sistema numerico binario:

- un **numero pari** termina sempre con un bit 0;
- un **numero dispari** termina sempre con un bit 1.

### ESEMPIO

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

$$\begin{array}{ccc} \text{Quozienti} & & \text{Resti} \\ \searrow & & \swarrow \\ 43 : 2 = & 21 & + 1 \text{ LSB (bit, } \text{meno} \text{ significativo)} \\ 21 : 2 = & 10 & + 1 \\ 10 : 2 = & 5 & + 0 \\ 5 : 2 = & 2 & + 1 \\ 2 : 2 = & 1 & + 0 \\ 1 : 2 = & 0 & + 1 \text{ LSB (bit, } \text{più} \text{ significativo)} \end{array}$$

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:

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



## Prova adesso!

- Conversione da decimale a binario

Converti da decimale a binario i seguenti numeri:

1 127

2 222

3 168

4 192

## ■ Conversione da decimale a esadecimale

L'algoritmo è il medesimo, dove ora il divisore è il numero 16, cioè la base del sistema di numerazione. Convertiamo lo stesso numero  $3157_{10}$ .

$$\begin{array}{ccc} \text{Divisore} & \text{Quoziente} & \text{Resto} \\ \searrow & \searrow & \searrow \\ 3157 : 16 = 197 + 5 \end{array}$$

e mettiamo i valori ottenuti in una tabella:

Iterazione	Valore	Divisore	Quozienti	Resti
1	3157	16	197	5

Ripetiamo il procedimento descritto in precedenza in modo da ottenere la seguente tabella:

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

Ricordiamo che nel sistema esadecimale A = 10, B = 11 , C = 12, D = 13 , E = 14 e F = 15

Ottenuto 0 come quoziente la divisione termina e prendendo i resti dall'ultimo al primo (dal basso verso l'alto) si ottiene:

$$N_8 = C\ 5\ 5$$

Quindi abbiamo completato la conversione ottenendo:

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

**ESEMPIO**

Vediamo un secondo esempio convertendo il numero  $(44157)_{10}$  in base 16:

$$44157 : 16 = 2759 + 13 \text{ (D)}$$

$$2759 : 16 = 172 + 7$$

$$172 : 16 = 10 + 12 \text{ (C)}$$

$$10 : 16 = 0 + 10 \text{ (A)}$$

I resti ottenuti sono 13 7 12 e 10 e “rigirando” tali valori si ottiene AC7D, che identifica il valore ottale del numero 44157.

Quindi il risultato è il seguente:

$$(44157)_{10} = (\text{AC7D})_{16}$$

**Prova adesso!**

- Conversione da decimale a esadecimale

Converti da decimale a esadecimale i seguenti numeri:

**1** 1271

**2** 2221

**3** 4681

**4** 18921

## ■ 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

- si evidenzia la parte intera del risultato, cioè se supera o meno l'unità;
- si sostituisce al numero la parte decimale del prodotto ottenuto;
- 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 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:

**A** 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:

$$(0.125)_{10} = (.001)_2$$

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 5 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	1.75	1
4	0.75	2	1.5	1
5	0.5	2	1.0	1

Quindi il risultato è il seguente:

$$(0.21875)_{10} = (.00111)_2$$

### ESEMPIO

Vediamo un ultimo esempio, convertendo il numero  $(0.45)_{10}$  con precisione 7 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	0.4	2	0.8	0
7	0.8	2	1.6	1

Il risultato della conversione è il seguente:

$$(.45)_{10} = (.0111001)_2$$

In questo caso la conversione è terminata solo per aver esaurito i 7 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 1100, con 01 come antiperiodo. La conversione completa sarebbe:

$$(.45)_{10} = (.0111001100110011001100\dots)_2$$

## 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	0.2	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  $0.\overline{1}_3$  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

### 1. Esercizi

**1** Converti i seguenti numeri decimali in binario

$$13_{10} = \underline{\hspace{2cm}}$$

$$11_{10} = \underline{\hspace{2cm}}$$

$$25_{10} = \underline{\hspace{2cm}}$$

$$21_{10} = \underline{\hspace{2cm}}$$

**2** Converti i seguenti numeri decimali in binario

$$10_{10} = \underline{\hspace{2cm}}$$

$$15_{10} = \underline{\hspace{2cm}}$$

$$29_{10} = \underline{\hspace{2cm}}$$

$$27_{10} = \underline{\hspace{2cm}}$$

$$33_{10} = \underline{\hspace{2cm}}$$

$$43_{10} = \underline{\hspace{2cm}}$$

**3** Converti i seguenti numeri decimali in binario

$$175_{10} = \underline{\hspace{2cm}}$$

$$195_{10} = \underline{\hspace{2cm}}$$

$$238_{10} = \underline{\hspace{2cm}}$$

$$241_{10} = \underline{\hspace{2cm}}$$

$$170_{10} = \underline{\hspace{2cm}}$$

$$203_{10} = \underline{\hspace{2cm}}$$

**4** Converti i seguenti numeri decimali in ottale

$$63_{10} = \underline{\hspace{2cm}}$$

$$302_{10} = \underline{\hspace{2cm}}$$

$$1197_{10} = \underline{\hspace{2cm}}$$

$$2421_{10} = \underline{\hspace{2cm}}$$

**5** Converti i seguenti numeri decimali in ottale

$$18_{10} = \underline{\hspace{2cm}}$$

$$83_{10} = \underline{\hspace{2cm}}$$

$$1123_{10} = \underline{\hspace{2cm}}$$

$$4681_{10} = \underline{\hspace{2cm}}$$

$$5349_{10} = \underline{\hspace{2cm}}$$

$$1007_{10} = \underline{\hspace{2cm}}$$

**6** Converti i seguenti numeri decimali in ottale

$$36_{10} = \underline{\hspace{2cm}}$$

$$246_{10} = \underline{\hspace{2cm}}$$

$$751_{10} = \underline{\hspace{2cm}}$$

$$8150_{10} = \underline{\hspace{2cm}}$$

$$10221_{10} = \underline{\hspace{2cm}}$$

$$19669_{10} = \underline{\hspace{2cm}}$$

**7** Converti i seguenti numeri decimali in esadecimale

$$17_{10} = \underline{\hspace{2cm}}$$

$$34_{10} = \underline{\hspace{2cm}}$$

$$51_{10} = \underline{\hspace{2cm}}$$

$$68_{10} = \underline{\hspace{2cm}}$$

**8** Converti i seguenti numeri decimali in esadecimale

$$17_{10} = \underline{\hspace{2cm}}$$

$$34_{10} = \underline{\hspace{2cm}}$$

$$51_{10} = \underline{\hspace{2cm}}$$

$$68_{10} = \underline{\hspace{2cm}}$$

**9** Converti i seguenti numeri decimali in esadecimale

$$171_{10} = \underline{\hspace{2cm}}$$

$$255_{10} = \underline{\hspace{2cm}}$$

$$301_{10} = \underline{\hspace{2cm}}$$

$$4660_{10} = \underline{\hspace{2cm}}$$

$$8840_{10} = \underline{\hspace{2cm}}$$

$$12705_{10} = \underline{\hspace{2cm}}$$

**10** Converti i seguenti numeri decimali in esadecimale

$$186_{10} = \underline{\hspace{2cm}}$$

$$238_{10} = \underline{\hspace{2cm}}$$

$$291_{10} = \underline{\hspace{2cm}}$$

$$2748_{10} = \underline{\hspace{2cm}}$$

$$9509_{10} = \underline{\hspace{2cm}}$$

$$15100_{10} = \underline{\hspace{2cm}}$$

## 2. Esercizi

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

$$\begin{array}{lcl} 1010.101_2 & = & \text{_____} \\ 1011.100_2 & = & \text{_____} \\ 11101.001_2 & = & \text{_____} \\ 10011.101_2 & = & \text{_____} \\ 110111.11_2 & = & \text{_____} \\ 111010.01_2 & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 11101.11010111 & = & \text{_____} \\ 10001.00001101 & = & \text{_____} \\ 00011.10011001 & = & \text{_____} \\ 11000.11111001 & = & \text{_____} \\ 10011.00100100 & = & \text{_____} \\ 10010.01001001 & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 341.1_5 & = & \text{_____} \\ 234.23_5 & = & \text{_____} \\ 1312.42_5 & = & \text{_____} \\ 2040.2_8 & = & \text{_____} \\ 2375.5_8 & = & \text{_____} \\ 1463.25_8 & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 268.1_{12} & = & \text{_____} \\ 568.4_{12} & = & \text{_____} \\ 543.51_{12} & = & \text{_____} \\ 123.2_{16} & = & \text{_____} \\ 222.5_{16} & = & \text{_____} \\ 2AB.19_{16} & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 23.466 & = & \text{_____} \\ 61.625 & = & \text{_____} \\ 13.543 & = & \text{_____} \\ 55.110 & = & \text{_____} \\ 19.999 & = & \text{_____} \\ 22.001 & = & \text{_____} \\ 41.700 & = & \text{_____} \end{array}$$

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

$$\begin{array}{l} N(2) = 10011.101_2 \\ N(2) = 110111.1111_2 \\ N(7) = 4625.32_7 \end{array}$$

$$\begin{array}{l} N(6) = 36541.32_6 \\ N(6) = 534.01_6 \end{array}$$

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

$$\begin{array}{lcl} 7516_{10} & = & \text{_____} \\ 108_{10} & = & \text{_____} \\ 108_{10} & = & \text{_____} \\ 115_{10} & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 0100100000010111 & = & \text{_____} \\ 1010101011110000 & = & \text{_____} \\ 0110011011000000 & = & \text{_____} \\ 1110100001010101 & = & \text{_____} \\ 0101010101011001 & = & \text{_____} \end{array}$$

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

$$\begin{array}{lcl} 5FD9 & = & \text{_____} \\ 610A & = & \text{_____} \\ ABCD & = & \text{_____} \\ 0100 & = & \text{_____} \\ CACA & = & \text{_____} \end{array}$$

**10** Effettua le seguenti conversioni:

$$\begin{array}{lcl} 54.75_{10} & = & \text{_____} \\ 123.625_{10} & = & \text{_____} \\ 123_{10} & = & \text{_____} \end{array}$$

**11** Effettua le seguenti conversioni:

$$\begin{array}{lcl} 103_5 & = & \text{_____} \\ 113_5 & = & \text{_____} \\ 213_5 & = & \text{_____} \\ 313_5 & = & \text{_____} \end{array}$$

**12** Effettua le seguenti conversioni:

$$\begin{array}{lcl} 23.32_{10} & = & \text{_____} \\ 21.12_{10} & = & \text{_____} \\ 54.75_{10} & = & \text{_____} \\ 75.54_{10} & = & \text{_____} \\ 132.231_{10} & = & \text{_____} \\ 231.132_{10} & = & \text{_____} \end{array}$$

**13** Effettua le seguenti conversioni:

$$\begin{array}{lcl} 111.111_2 & = & \text{_____} \\ 101.010_2 & = & \text{_____} \\ 10.1101_2 & = & \text{_____} \\ 1.10101_2 & = & \text{_____} \\ 0.100110_2 & = & \text{_____} \\ 0.110011001_2 & = & \text{_____} \end{array}$$

# Conversione tra le basi binarie

In questa lezione 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 lezione 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 fare due osservazioni:

- la **codifica ottale** coincide con i primi otto elementi della **codifica decimale**;
- 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 Conversione di 110111 in ottale

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

$$110_2 \quad 111_2$$

Convertiamo in ottale ogni gruppo di 3 cifre:

$$110_2 = 6_8 \quad \text{e} \quad 111_2 = 7_8$$

Quindi il numero ottale risultante è il seguente:

$$110111_2 = 67_8$$

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

$110$	$111_2$
V	V
6	$7_8$

### ESEMPIO Conversione di 101100 in ottale

Vediamo un secondo esempio, dove convertiamo il numero  $101100_2$ .

Schematicamente abbiamola situazione mostrata a lato:

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

$101$	$100$
V	V
5	4



## Prova adesso!

Effettua la conversione in ottale dei seguenti numeri binari:

1 110010

3 110010

2 101001

4 100000

### ESEMPIO Conversione di 11001010 in ottale

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  $11001010_2 = 312_8$ .

$011$	$001$	$010$
V	V	V
3	1	2

**ESEMPIO**

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: ►

$001 \quad 011 \quad 010 \ . \ 110$   
  
 1      3      2    .    6

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  $1011010.11_2$  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  $1011010.11$  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 Conversione di  $63_8$  in binario**

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

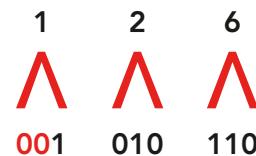
$6 \quad 3$   
  
 110    011

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

**ESEMPIO** *Conversione di  $126_8$  in binario*

Convertiamo il numero ottale  $126_8$  in binario:

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

**Prova adesso!**

Effettua la conversione in binario dei seguenti numeri:

- 1**  $765_8$
- 2**  $432_8$
- 3**  $272_8$
- 4**  $123_8$

- 5**  $12.34_8$
- 6**  $67.56_8$
- 7**  $98.345_8$
- 8**  $123.567_8$

**■ Conversione tra binari ed esadecimale**

Il sistema esadecimale si basa sulla numerazione in base 16, quindi con  $b = 16$ , che utilizza i simboli  $\Sigma = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ , 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

**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 Conversione di 10101110 in esadecimale

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:

$$1010_2 = A_h \text{ e } 1110_2 = E_h$$

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:



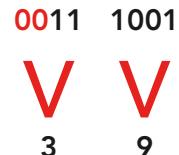
### ESEMPIO Conversione di 111001 in esadecimale

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 Conversione di 111010.1 in esadecimale

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.



## Prova adesso!

Effettua la conversione in esadecimale dei seguenti numeri:

- |          |              |          |              |
|----------|--------------|----------|--------------|
| <b>1</b> | $11011010_2$ | <b>5</b> | 10101.01     |
| <b>2</b> | $111110_2$   | <b>6</b> | 11001.101    |
| <b>3</b> | $110000_2$   | <b>7</b> | 101001.0111  |
| <b>4</b> | $11011_2$    | <b>8</b> | 1010101.0101 |

Come per la conversione da binario, fate attenzione quando aggiungete gli zeri e raggruppate i nibble: gli zeri si inseriscono sempre a sinistra.

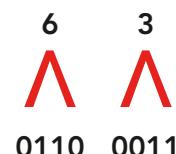
### Da esadecimale a binario

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

#### ESEMPIO **Conversione di $63_h$ in binario**

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

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

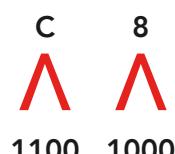


Vediamo un secondo esempio.

#### ESEMPIO **Conversione di $C8_h$ in binario**

Convertiamo il numero esadecimale  $C8_h$  in binario:

Il risultato è  $C8_h = 1100\ 1000_2$ .



#### ESEMPIO

Come ultimo esempio convertiamo il numero  $1F.4_h$  in binario:  
Il risultato è  $1F.4_h = 0001\ 1111.0100_2$  che, dopo aver eliminato gli zeri “superflui”, diviene:

$$1F.4_h = 11111.01_2$$





## Prova adesso!

Effettua la conversione in binario dei seguenti numeri:

**1**  $1A_h$

**3**  $3C_h$

**5**  $7B.A$

**7**  $123.AB$

**9**  $4E.34$

**2**  $2B_h$

**4**  $F4_h$

**6**  $8E.C$

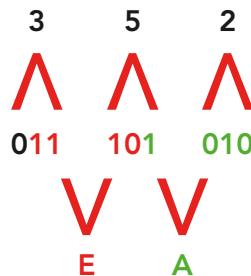
**8**  $585.EF$

## ■ 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 *Conversione di $352_8$ in esadecimale*

Convertiamo  $(352)_8$  in esadecimale:



Quindi il risultato è  $(352)_8 = (11 101 010)_2$  da cui si ottiene  $(352)_8 = (1110 1010)_2 = (EA)_h$ .



## Prova adesso!

Effettua la conversione in esadecimale dei seguenti numeri ottali:

**1**  $172_8$

**3**  $475_8$

**5**  $544_8$

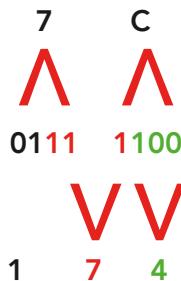
**2**  $632_8$

**4**  $303_8$

**6**  $765_8$

### ESEMPIO *Conversione di $7C_h$ in ottale*

Convertiamo  $(7C)_h$  in ottale:

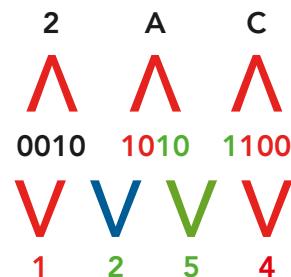


Quindi  $(7C)_h = (0111 1100)_2$  da cui si ottiene  $(7C)_h = (001 111 100)_2 = (174)_8$ .

**ESEMPIO** *Conversione di  $(2AC)_h$  in ottale*

Convertiamo  $(2AC)_h$  in ottale:

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

*Prova adesso!*

Effettua la conversione in ottale dei seguenti numeri esadecimali:

- |  |  |
|--|--|
| <b>1</b> $74_h$<br><b>2</b> $C3_h$<br><b>3</b> $19A_h$ | <b>4</b> $73D_h$<br><b>5</b> $ABC_h$<br><b>6</b> $FE1_h$ |
|--|--|

## Verifichiamo le competenze

### 1. Esercizi

**1** Esegui le seguenti conversioni da binario a ottale

$$010100_2 = \underline{\hspace{2cm}}$$

$$000111_2 = \underline{\hspace{2cm}}$$

$$101010_2 = \underline{\hspace{2cm}}$$

$$001110_2 = \underline{\hspace{2cm}}$$

**2** Esegui le seguenti conversioni da binario a ottale

$$110011_2 = \underline{\hspace{2cm}}$$

$$100001_2 = \underline{\hspace{2cm}}$$

$$1110011_2 = \underline{\hspace{2cm}}$$

$$1100001_2 = \underline{\hspace{2cm}}$$

$$10110011_2 = \underline{\hspace{2cm}}$$

$$11000101_2 = \underline{\hspace{2cm}}$$

**3** Esegui le seguenti conversioni da ottale a binario

$$12_8 = \underline{\hspace{2cm}}$$

$$34_8 = \underline{\hspace{2cm}}$$

$$567_8 = \underline{\hspace{2cm}}$$

$$10_8 = \underline{\hspace{2cm}}$$

**4** Esegui le seguenti conversioni da ottale a binario

$$21_8 = \underline{\hspace{2cm}}$$

$$303_8 = \underline{\hspace{2cm}}$$

$$200_8 = \underline{\hspace{2cm}}$$

$$1753_8 = \underline{\hspace{2cm}}$$

$$2542_8 = \underline{\hspace{2cm}}$$

$$5104_8 = \underline{\hspace{2cm}}$$

**5** Esegui le seguenti conversioni da binario a esadecimale

$$1111010_2 = \underline{\hspace{2cm}}$$

$$10101110_2 = \underline{\hspace{2cm}}$$

$$11111011_2 = \underline{\hspace{2cm}}$$

$$11010101_2 = \underline{\hspace{2cm}}$$

**6** Esegui le seguenti conversioni da binario a esadecimale

$$111010_2 = \underline{\hspace{2cm}}$$

$$10111110_2 = \underline{\hspace{2cm}}$$

$$1011011_2 = \underline{\hspace{2cm}}$$

$$10010001_2 = \underline{\hspace{2cm}}$$

$$111011011_2 = \underline{\hspace{2cm}}$$

$$10101010001_2 = \underline{\hspace{2cm}}$$

**7** Esegui le seguenti conversioni da esadecimale a binario

$$A7F_{16} = \underline{\hspace{2cm}}$$

$$B4C_{16} = \underline{\hspace{2cm}}$$

$$27E_{16} = \underline{\hspace{2cm}}$$

$$490_{16} = \underline{\hspace{2cm}}$$

**8** Esegui le seguenti conversioni da esadecimale a binario

$$322 = \underline{\hspace{2cm}}$$

$$A31 = \underline{\hspace{2cm}}$$

$$F2B = \underline{\hspace{2cm}}$$

$$7764 = \underline{\hspace{2cm}}$$

$$2359 = \underline{\hspace{2cm}}$$

$$4B38 = \underline{\hspace{2cm}}$$

**9** Esegui le seguenti conversioni da ottale a esadecimale

$$72_8 = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}} h$$

$$54_8 = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}} h$$

$$321_8 = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}} h$$

$$610_8 = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}} h$$

**10** Esegui le seguenti conversioni da esadecimale a ottale

$$27_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$18_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$AB2_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$290_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$345_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$67FE_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$CD63_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$279B_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$C163_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

$$E279_h = \underline{\hspace{2cm}}_2 = \underline{\hspace{2cm}}_8$$

## AREA digitale



Esercizi per il recupero / Esercizi per l'approfondimento



# Immagini, suoni e filmati

**In questa lezione impareremo...**

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

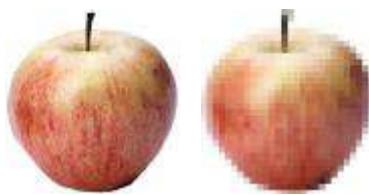
## ■ Introduzione

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 lezione verranno analizzati i formati digitali più utilizzati per le componenti multimediali che in base alla loro tipologia possono essere così classificate:

- **immagini scalari** o raster, utilizzate per le fotografie, le scansioni, le immagini biomediche ecc., dove un'immagine rappresenta una scena con una matrice di numeri (**bitmapped**);
- **immagini vettoriali**, utilizzate principalmente nel disegno geometrico (CAD), nei loghi, nelle illustrazioni, nelle mappe, nei grafici, dove una scena ha una **descrizione matematica di forme e linee**.

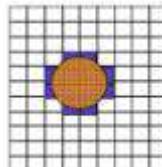
## ■ Immagini digitali

Nella rappresentazione digitale di **immagini** sia pittoriche che grafiche è necessario trasformare l'**immagine analogica** (per esempio una fotografia), cioè un **insieme continuo** di informazioni (luce, colore) in due dimensioni, in un insieme di parti distinte che possono essere codificate separatamente come numeri (**discretizzazione**):

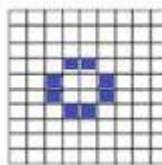
- la prima fase del processo di digitalizzazione avviene sovrapponendo idealmente una griglia fittissima di minuscole celle denominate **pixel** (picture element): questa operazione si chiama **campionamento**;
- successivamente vengono codificati i pixel associando a ogni punto un numero e tale numero corrisponde, mediante una tabella di corrispondenza (**tavolozza**), a colori diversi o a sfumature diverse di un particolare colore: questa operazione si chiama **quantizzazione**.

### ESEMPIO

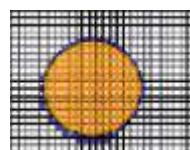
Vediamo come rappresentare una circonferenza: la sovrapponiamo a una griglia come in figura:



e ne otteniamo i pixel che la individuano

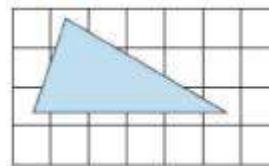
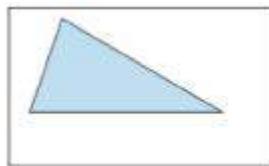


Maggiore è la "rete di cellette", cioè più piccola è la dimensione della celletta, maggiore è il numero di pixel che viene utilizzato per **definire** l'immagine: chiamiamo **risoluzione** dell'immagine la dimensione della griglia usata per il campionamento dell'immagine che è in diretta relazione con la qualità con cui si rappresenta l'immagine (valori tipici sono  $640 \times 480$ ,  $1024 \times 768$  ecc.).



## Bianco e nero

Vediamo per esempio come è possibile digitalizzare un triangolo in bianco e nero (e successivamente aggiungeremo i colori) rappresentato nelle figure seguenti:



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 destra.

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.

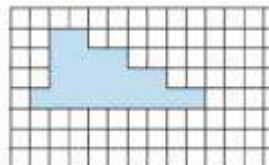
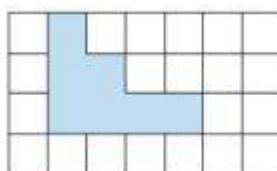
0	1	0	0	0	0	0
25	25	24	25	26	27	28
0	1	1	0	0	0	0
15	16	17	16	17	18	19
0	1	1	1	1	0	0
8	9	10	11	12	13	14
0	0	0	0	0	0	0

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 avvicinare 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 destra:

- nella prima codifica abbiamo rappresentato l'immagine con  $7 \times 4 = 28$  bit;
- nella seconda codifica abbiamo abbiammo una griglia composta da  $14 \times 8 = 112$  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.

Anche per le fotocamere digitali spesso viene utilizzato come parametro di riferimento la risoluzione anche se non sempre è il fattore determinante per la qualità dell'immagine prodotta; in generale si usano multipli dei **pixel** come il **Megapixel** (= 1 milione di pixel).

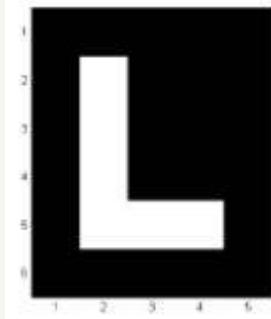


### Prova adesso!

Attribuendo come valore del pixel:

- ▶ 1 per rappresentare i pixel bianchi,
  - ▶ 0 per rappresentare i pixel neri,
- codifica l'immagine a lato ▶

- Rappresentazione in bianco/nero



### 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**).

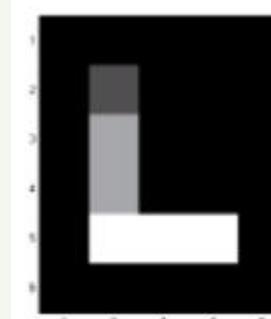


### Prova adesso!

Utilizzando quattro livelli di grigio

- ▶ 00 = 0 (nero),
  - ▶ 01 = 1 (grigio scuro),
  - ▶ 10 = 2 (grigio chiaro),
  - ▶ 11 = 3 (bianco),
- codifica l'immagine a lato ▶

- Rappresentazione in toni di grigio



Generalmente per le immagini in scala di grigio, è sufficiente utilizzare 8 bit per pixel dato che il sistema visivo umano difficilmente è in grado di distinguere un numero maggiore di livelli di grigio: nelle applicazioni mediche e/o professionali si utilizzano generalmente un numero maggiore di bit per pixel, da 10 a 12 bit per pixel.

### ESEMPIO

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

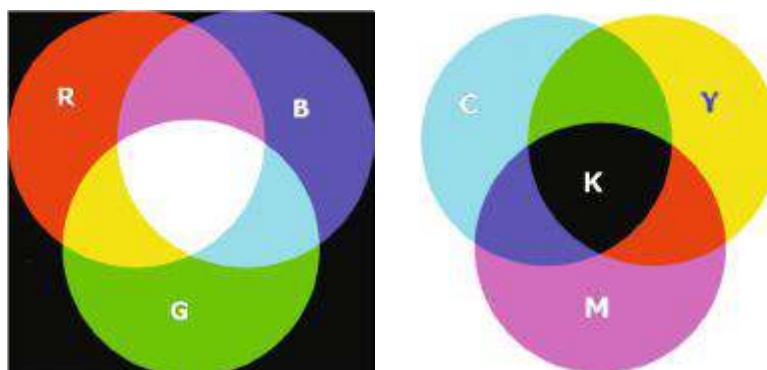
## Immagini a colori

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 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:



Nel metodo **RGB** si associano 256 livelli a ciascun colore, quindi si utilizzano 8 bit per colore, per un totale di 24 bit:

- 8 bit (1 byte) per il rosso [0-255];
- 8 bit (1 byte) per il verde [0-255];
- 8 bit (1 byte) per il blu [0-255].

In questo modo è possibile produrre fino a  $2^{24} = 16.777.216$  milioni di colori sullo schermo

### ESEMPIO

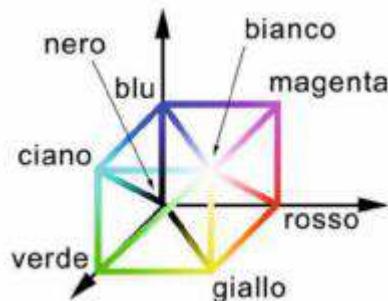
Quando il valore di tutte le componenti è 255, si ottiene il bianco puro, mentre quando il valore di tutte le componenti è 0, si ottiene il nero puro.

il bianco è dato da 255, 255, 255



Il giallo è dato da 255, 255, 0

Possiamo rappresentare nello spazio a tre dimensioni le possibili combinazioni delle tre componenti, come riportato nella figura seguente:



### AREA digitale



Tabella dei colori RGB



### CODIFICA BITMAP

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

### ESEMPIO

Vediamo una fotografia in formato **RGB** scomposta nelle sue tre componenti nei singoli colori primari:



Utilizziamo ad esempio un programma come **Gimp** oppure **Photoshop** che permette la scomposizione in livelli, cioè permette di separare le componenti in nuove immagini in scala di grigi in cui ogni livello rappresenta uno dei canali.

Componente rosso R



Componente verde G



Componente blu B



## Palette di colori

Un'immagine può occupare molto spazio anche se non tutti i 16.7 milioni di colori sono usati contemporaneamente: in queste situazioni si può usare un sottoinsieme dei colori rappresentandoli su una **tavolozza di colori** (teoricamente detta **palette** ), assegnare a essi un numero e utilizzare questo per poi codificare i bit dell'immagine in riferimento al colore della tavolozza.



Con **palette** si intende la gamma di colori in dotazione che può essere quella della scheda grafica di un computer oppure quelli disponibili per realizzare un disegno o una fotografia.

### ESEMPIO

Supponiamo ad esempio di utilizzare solo 4 colori RGB: costruiremo una tavolozza come in figura:

Ogni colore della tavolozza può essere rappresentato con due soli bit, quindi nel caso del seguente disegno composto da una immagine  $3 \times 3$  pixel la rappresentazione sarebbe:

### RGB dei colori

1	81 12 D4	00
2	44 D6 D5	01
3	3E 52 18	10
4	1B BC AA	11

	<b>81 12 D4</b> pixel 7 <b>81 12 D4</b> pixel 4 <b>3E 52 18</b> pixel 1	<b>44 D6 D5</b> pixel 8 <b>44 D6 D5</b> pixel 5 <b>1B BC AA</b> pixel 2	<b>3E 52 18</b> pixel 9 <b>44 D6 D5</b> pixel 6 <b>1B BC AA</b> pixel 3	00 01 10 00 01 01 10 11 11
--	--	--	--	----------------------------------

Senza l'utilizzo della **palette** si sarebbero utilizzati  $3 \text{ byte} \times 9 = 27 \text{ byte} = 216 \text{ bit}$ .

Con l'utilizzo della **palette** l'immagine occupa solamente 18 bit.

Il caso tipico è quello di utilizzare una paletta di 256 colori, e quindi di avere una sua indicizzazione con un insieme di 8 bit: in questo modo si riduce l'occupazione di memoria a 1/3, dato che invece che utilizzare 3 byte per colore se ne utilizza uno soltanto.

Dobbiamo ricordare che uno spazio aggiuntivo è necessario per memorizzare la **palette**: nel caso di 256 colori di palette abbiamo un ulteriore  $256 \times 3 = 768 \text{ byte}$  di spazio occupato, che comunque risulta essere trascurabile soprattutto con immagini di grosse dimensioni.

## Risoluzione di una immagine: PPI e DPI



### RISOLUZIONE

Definiamo **risoluzione** la qualità con la quale una immagine è rappresentata.

La qualità di una immagine è legata al numero di pixel che vengono utilizzati per la sua rappresentazione, al numero di colori e alla sua dimensione.

Come unità di misura della risoluzione viene utilizzato il **pixel**, cioè il numero di punti che vengono utilizzati per rappresentarla, e si utilizzano due diversi riferimenti a seconda se si tratta di immagini a video oppure immagini stampate:

- a video si utilizzano i **PPI**, cioè **Pixel Per Inches**;
- in stampa si utilizzano **DPI**, cioè **Dot Per Inches**.

Sono sempre "punti per pollice", ma a video il punto è il **pixel** mentre in stampa è il punto impresso sul foglio. Ricordiamo che un pollice corrisponde a 2.54 cm e, quindi, 1 cm è equivalente a 0.39370 pollici.

Nella successiva immagine vediamo la fotografia di un'aquila che utilizza 256 toni di grigio ma con quattro diverse definizioni.



La **risoluzione** influenza la **grandezza** dell'immagine:

- a parità di pixel, aumentando la risoluzione diminuisce la grandezza;
- a parità di grandezza, aumentando la risoluzione, aumentano i pixel.

Quindi possiamo affermare che **risoluzione** e **grandezza** sono **inversamente proporzionali**.

Vediamo in una tabella la relazione tra occupazione di memoria, grandezza e risoluzioni di alcune situazioni tipiche.

Risoluzione	Dimensione		Occupazione di memoria (Bytes)	
	Pollici (in) cm	pixel	256 grigi	Colori RGB
75 PPI	6×4 15×10	450×300	135.000	405.000
150 PPI	6×4 15×10	900×600	540.000	1.620.000
300 PPI	6×4 15×10	1800×1200	2.160.000	6.480.000
600 PPI	6×4 15×10	3600×2400	8.640.000	25.920.000

**ESEMPIO**

Vediamo ora due esempi:

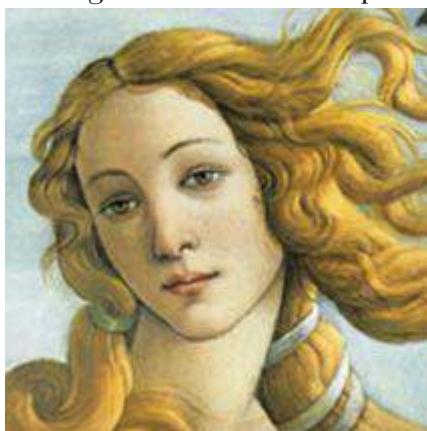
- nel primo **a parità di pixel** aumentiamo la risoluzione da 72 PPI a 320 PPI e di conseguenza diminuisce la grandezza;



Dimensione pixel: 300 KB	
Larghezza:	320 pixel
Altezza:	320 pixel
Dimensioni documento	
Larghezza:	11,29 cm
Altezza:	11,29 cm
Risoluzione:	72 pixel/pollice

Dimensione pixel: 300 KB	
Larghezza:	320 pixel
Altezza:	320 pixel
Dimensioni documento	
Larghezza:	9,84 cm
Altezza:	2,54 cm
Risoluzione:	320 pixel/pollice

- nel secondo **a parità di grandezza**, aumentando la risoluzione da 36 PPI a 150 PPI di conseguenza aumentano i pixel e il peso.



Dimensione pixel: 75 KB	
Larghezza:	160 pixel
Altezza:	160 pixel
Dimensioni documento	
Larghezza:	11,29 cm
Altezza:	11,29 cm
Risoluzione:	36 pixel/pollice

Dimensione pixel: 1,27 MB	
Larghezza:	667 pixel
Altezza:	667 pixel
Dimensioni documento	
Larghezza:	11,29 cm
Altezza:	11,29 cm
Risoluzione:	150 pixel/pollice



## Prova adesso!

- Risoluzione di una immagine
- Definizione di una immagine

Scegli una immagine in tonalità di grigio e con un programma di elaborazione grafica (come Photofiltre, Gimp o Photoshop) e cambia per tre volte la risoluzione mantenendo inalterata la dimensione. Quindi a partire dalla medesima immagine, modifica 6 volte la dimensione.

Riporta in una tabella la risoluzione scelta e la dimensione dell'immagine e il peso in memoria del file.

Prova infine a stampare tutte le immagini e confrontale tra di loro.

Un discorso più complesso deve essere fatto per la **stampa delle immagini**, ma entrare nel dettaglio esula dagli scopi di questa trattazione introduttiva.

Possiamo comunque osservare che in stampa una fotografia  $15 \times 10$  cm dovrà essere stampata almeno alla risoluzione di 280 **DPI**, altrimenti risulta molto “sgranata” e normalmente le riviste utilizzano una risoluzione minima di 300 **DPI** per stampare le loro fotografie.

La tabella seguente riporta alcuni valori di risoluzione necessari per ottenere una stampa di buona qualità nei diversi formati.

Formato	Misure in cm	DPI	PIXEL
A4	$29.7 \times 21$	300	$3508 \times 2480$
A3	$42 \times 29.7$	250	$4134 \times 2923$
A2	$59.4 \times 42$	200	$4677 \times 3307$
A1	$84.1 \times 59.4$	150	$4967 \times 3508$
A0	$118.9 \times 84.1$	110	$5149 \times 3642$
Poster 1	$300 \times 200$	45	$5315 \times 3543$
Poster 2	$500 \times 300$	30	$5906 \times 3543$

Si può osservare come all'aumentare della dimensione di stampa non vengono raddoppiati i pixel necessari, dato che più grande è l'immagine, minore è la densità di pixel di cui abbiamo bisogno, dato che molto presumibilmente queste immagini verranno osservate da una distanza maggiore e quindi l'occhio umano aumentando la distanza non percepisce la minor densità di punti.

Possiamo fare il discorso “a rovescio”, cioè avendo una immagine digitale e volendo scegliere la dimensione ottimale di stampa, otteniamo la seguente tabella:

MEGAPIXEL	RISOLUZIONE	STAMPA a 72 dpi	STAMPA a 300 dpi
1 Megapixel	$1280 \times 768$	$45 \times 27$	$10 \times 6$
3 Megapixel	$2048 \times 1536$	$72 \times 54$	$17 \times 13$
5 Megapixel	$2560 \times 1920$	$90 \times 67$	$21 \times 16$
11 Megapixel	$4064 \times 2704$	$143 \times 95$	$34 \times 22$



## Zoom su...

### BASSA RISOLUZIONE

La distinzione tra alta risoluzione e bassa risoluzione ha assunto anche rilevanza giuridica in Italia perché la legge 9 gennaio 2008, n. 2, integrando l'articolo 70 della legge n. 633/41 sul diritto d'autore, ha previsto per i siti non lucrativi, la possibilità di riprodurre, esclusivamente a bassa risoluzione, immagini per fini didattici o scientifici. La legge però manca di dare un'esatta definizione di che cosa debba intendersi per bassa risoluzione, il che porta a gravi e seri problemi interpretativi di tale legge. (fonte: Wikipedia)

## ■ Compressione delle immagini

Come abbiamo visto è abbastanza semplice calcolare l'occupazione su disco di un'immagine.

Vediamo due casi, uno senza e l'altro con l'utilizzo di una **palette**:

- 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 ( $256 \times 3 = 768$  byte) e 8 bit per codificare ogni pixel, quindi otteniamo per un'immagine di **640 × 480** pixel una dimensione di  $(640 \times 480 \times 1) + (768 \times 1) = 307.200 + 768$  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.

Questa tecnica è utilizzata 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.

### AREA digitale



Immagini raster nei diversi formati e dimensioni



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)**.

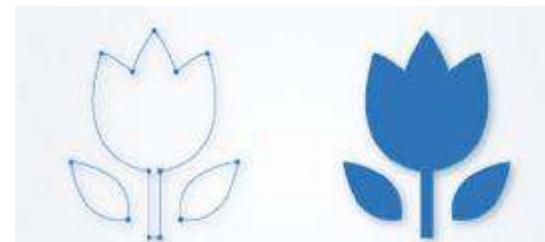
## ■ Immagine vettoriale

Una particolare rappresentazione delle immagini è quella **vettoriale (vector)**: 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.

### ESEMPIO

Per esempio, l'immagine a fianco viene costruita a partire da punti e dal loro congiungimento mediante funzioni matematiche: non vengono memorizzati cioè tutti i **pixel**, ma solo quelli evidenziati.

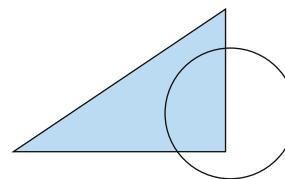
In questo modo le immagini risultano essere svincolate dalla risoluzione e possono essere utilizzate con qualsiasi dimensione senza alcuna perdita di qualità o aumento di peso.



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. ►

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.

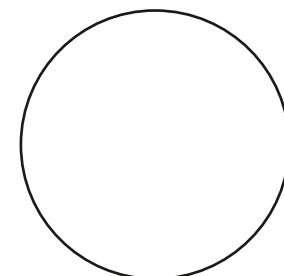
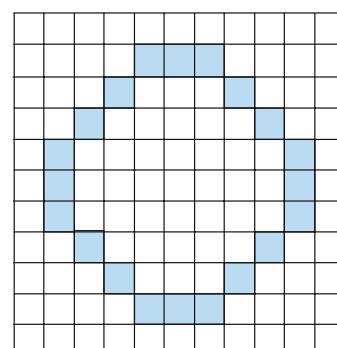


circle 98 66 50  
polyline 0 48 88 152 88 48

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

Questo 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. ►



Riassumendo possiamo individuare le principali differenze tra la modalità **raster** e quella **vector** in:

- **dimensione**: le immagini **raster** generalmente occupano più memoria di quelle **vector**;
- **scalabilità**: le vettoriali possono essere scalate senza perdita di qualità, le immagini **raster** no;
- **caratteristiche visuali**:
  - **vector** forme semplici con aree a colori piatti;
  - **raster** forme complesse con aree a colori sfumati;
- **conversione**: rasterizzare le immagini **vector** è diretto, non il contrario!

## AREA digitale



Immagini vettoriali

## ■ 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 conseguono 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 celluloido.



### FOTOGRAFMA

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 moviele 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.

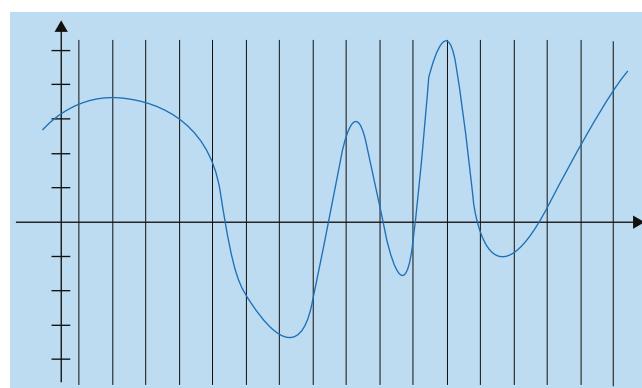
Infine incide anche la qualità del **commento sonoro** che viene codificato come un file audio e “sincronizzato” alle immagini.

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.

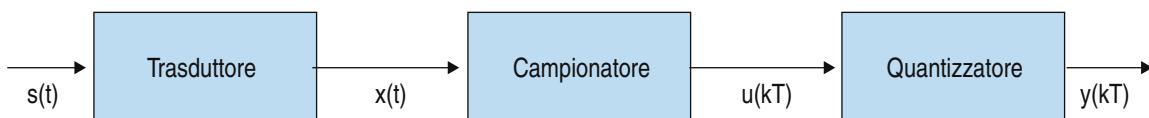
## ■ Suoni digitali

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.



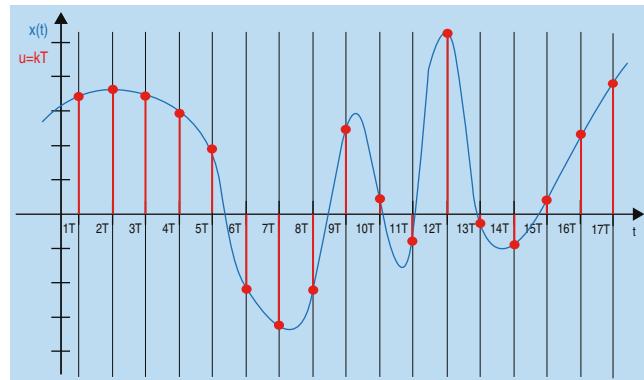
In generale, per tradurre un qualunque segnale analogico nel corrispettivo digitale sono necessari i seguenti passaggi:



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.

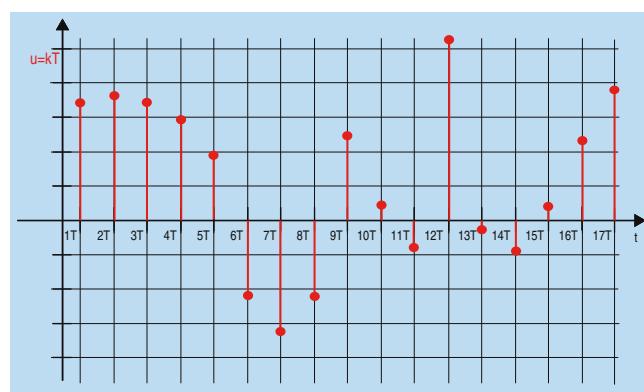
$$f_c = \frac{1}{T}$$



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 a lato. ►

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 "lettura" 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 compact 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:  $16 \times 88.200 = 1.411.200$  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  $1.4 \times 3600 = 5040$  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 Novanta 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 converte i file musicali **WAV** in **MIDI**, ma sono abili strumentisti (*midifilist*) 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  $1920 \times 1080$  pixel in true color: quindi per ogni secondo servono  $1920 \times 1080 \times 3 \times 24 \sim 149$  Mbyte (bitrate  $149 \times 8 \sim 1.2$  Gbit/s), per un’ora otteniamo  $149 \times 3600 =$  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  $24 \times 96.000 \times 8 \cong 18.4$  Mbit (bitrate 18.4 Mbit/s), per un’ora di audio servono quindi  $18.4/8 \times 3600 \cong 8$  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

### 1. Completamento

Completa le frasi seguenti scegliendo tra le parole proposte.

- 1 Il colore viene ottenuto da almeno tre colori di base, secondo un modello cromatico:

..... (....., ....., .....) )

oppure da quattro colori base:

..... (....., ....., ....., .....) )

RGB • verde • giallo • ciano • nero • rosso • blu • magenta • CMYK

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

24 • 4 • 40

- 3 Un CD musicale di ..... MB contiene generalmente circa ..... minuti di musica in formato WAV e ..... ore in formato Mp3.

700 • 12 • 70

### 2. Vero o 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

### 1. 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  $m \times n$  bit, ovvero  
 $\underline{\quad} \times \underline{\quad}$  circa =  $\underline{\quad}$  byte

Il bitrate minimo necessario per trasferire quest'immagine in un secondo è  
 $\underline{\quad} / 1 = \underline{\quad}$  bit/s

#### b) scala di grigi

Lo spazio richiesto per memorizzare un'immagine salvata in bianco e nero è  $m \times n$  byte:  
 $\underline{\quad} \times \underline{\quad}$  circa =  $\underline{\quad}$  byte

Il bitrate minimo necessario per trasferire quest'immagine in quattro secondi è  
 $\underline{\quad} / 4 = \underline{\quad}$  bit/s

#### c) true color

Lo spazio richiesto per memorizzare un'immagine salvata in true color è  
 $m \times n \times \underline{\quad}$  byte, ovvero  $\underline{\quad} \times \underline{\quad} \times \underline{\quad}$  circa =  $\underline{\quad}$  byte

Il bitrate minimo necessario per trasferire quest'immagine in due secondi è  
 $\underline{\quad} / 2 = \underline{\quad}$  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  $\underline{\quad}$  byte + per l'immagine  $\underline{\quad}$  byte =  $\underline{\quad}$  byte

Il bitrate minimo necessario per trasferire quest'immagine in tre secondi è  
 $\underline{\quad} / 3 = \underline{\quad}$  bit/s

## 2. Esercizi

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

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

# ESERCITAZIONI DI LABORATORIO 1

## LA COMPRESSIONE DEI DATI CON HUFFMAN

### ■ Generalità

In generale il termine “compressione” porta a pensare all’operazione meccanica che permette di ridurre una o più caratteristiche geometriche di un corpo, come ad esempio la compressione di una lattina o di una scatola vuota: anche in informatica assume lo stesso significato, cioè *una operazione che permette di ridurre la dimensione di un file*.

Dato che un file è composto da una sequenza di bit e non è possibile ridurre la dimensione dei singoli bit è necessario effettuare un altro tipo di operazione, cioè ridurre il numero di bit ma in modo tale da mantenere invariata l’informazione originale.

La compressione si rende necessaria soprattutto nelle operazioni di trasmissione dei file dato che ancora oggi la maggior parte delle linee di comunicazione non garantiscono una adeguata velocità trasmissiva e, quindi, è necessario ridurre la dimensione degli oggetti da trasmettere in modo da risparmiare tempo e risorse!

Come vedremo in seguito, a differenza dei file musicali e multimediali in genere dove è anche possibile perdere una parte di messaggio senza “che i sensi umani” se ne accorgano, nel caso di trasmissione di file di testo è necessario che quanto compresso rimanga identico una volta che viene effettuata l’operazione opposta, cioè quella di decompressione.

Un metodo di compressione è quello che prende il nome di **compressione statistica** che, come suggerito dal nome, si basa sullo studio statistico dei dati da comprimere in modo da determinare la frequenza con la quale ciascun carattere compare nelle parole e di sfruttare tale analisi per effettuare una codifica del testo.

L’idea alla base di questi algoritmi è quella di utilizzare un codice a lunghezza variabile, codificando cioè con pochi bit le lettere che si ripetono con maggior frequenza e di aumentare la lunghezza man mano che la loro frequenza diminuisce all’interno del messaggio.

## ■ La codifica di Huffman

L'algoritmo di **Huffman** è un algoritmo che sfrutta la compressione statistica ed è un buon algoritmo per la compressione dei dati: offre le migliori performance soprattutto per i file di testo.

Studiamo il funzionamento mediante un esempio.

### ESEMPIO

Vogliamo codificare la stringa:

**CARAMELLA**

Per prima cosa contiamo le frequenze dei caratteri, ottenendo:

Carattere	Frequenza
A	3
L	2
C	1
R	1
M	1
E	1

Codificando la parola in ASCII otterremmo una stringa lunga 9 byte, cioè  $9 \times 8 = 72$  bit. Se invece riuscissimo a codificare le lettere con frequenza maggiore con un numero di bit ridotto, ad esempio codificassimo A e L con soli 2 bit e le altre con 3 bit, la lunghezza totale sarebbe  $5 \times 2 + 4 \times 3 = 22$  bit.

Otterremmo in questo modo una riduzione della lunghezza di oltre il 70%.

L'algoritmo di **Huffman** applica proprio questa strategia ottenendo ottime percentuali di riduzione soprattutto all'aumentare della lunghezza del messaggio da comprimere.

Per poter definire la codifica dei singoli caratteri, la parte centrale dell'algoritmo effettua la costruzione di una particolare struttura (chiamata in informatica **albero binario**) generandola in base al numero di occorrenze di ogni carattere nel testo da codificare: alla fine della elaborazione i caratteri maggiormente ripetuti sono posizionati vicino alla parte iniziale della struttura (**radice dell'albero**) mentre gli altri caratteri saranno via via più distanti in base alla loro frequenza.

Si creano dei “percorsi” dove ogni “tappa intermedia” aggiunge un bit alla codifica: i percorsi più corti vengono trasformati quindi con pochi bit mentre quelli più lunghi richiederanno codifiche più lunghe.

La realizzazione informatica degli **alberi** rientra nei programmi del quarto anno di corso ma la comprensione di come viene utilizzato l'**albero** per effettuare la codifica può essere fatta anche senza scrivere il codice in linguaggio di programmazione ma semplicemente simularlo sulla carta.

Effettuiamo la codifica di **Huffman** proseguendo col semplice esempio di codifica della parola CARAMELLA, spiegando i vari passi per arrivare alla codifica dei singoli caratteri.

Disponiamo i caratteri in una tabella disponendoli in ordine crescente di frequenza.

C	R	M	E	L	A
1	1	1	1	2	3

Come prima operazione per la costruzione della struttura (**l'albero**) prendiamo i primi due elementi che sono tra quelli che si ripetono con minor frequenza: vengono inseriti in un disegno come quello riportato nella figura seguente collegandoli a un elemento intermedio (**nodo**) che al suo interno contiene un numero ottenuto dalla somma delle frequenze dei due caratteri.

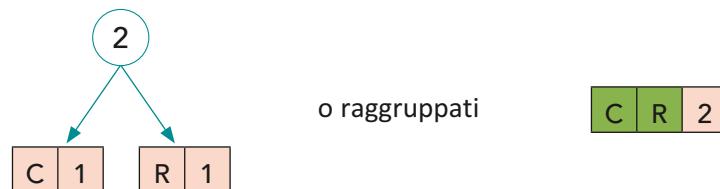


figura 1

Riordiniamo la tabella delle frequenze sostituendo il nuovo elemento così costruito:

M	E	L	C-R	A
1	1	2	2	3

Ripetiamo la costruzione come fatta in precedenza partendo sempre dai primi due elementi che vengono inseriti con un nodo avente peso la somma.

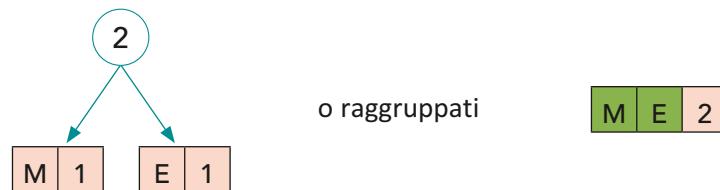


figura 2

Riordiniamo la tabella delle frequenze sostituendo il nuovo elemento così costruito:

L	C-R	M-E	A
2	2	2	3

Ripetiamo la costruzione partendo sempre da due elementi con frequenza minore: primi due elementi che vengono inseriti come figli di un nodo avente peso la loro somma e il nuovo albero ottenuto si è alzato di un livello:

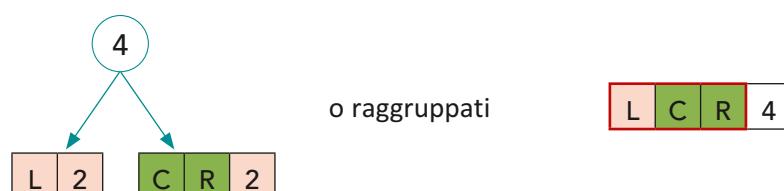


figura 3

Oppure sostituendo la figura 1 estesa otteniamo:

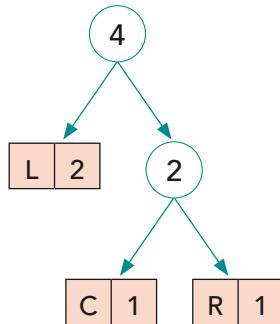


figura 4

che è una struttura su “due livelli“.

Ripetiamo il procedimento e ordiniamo i nuovi elementi:

M-E	A	L-C-R
2	3	4

Raggruppiamo sempre i primi due elementi ottenendo il seguente disegno:

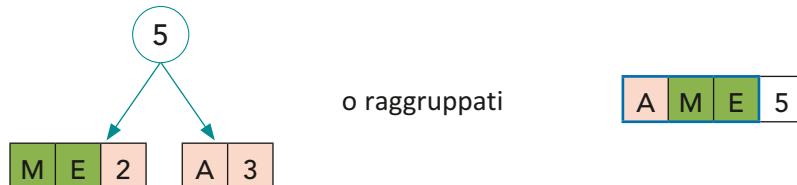


figura 5

Oppure sostituendo la figura 2 estesa otteniamo:

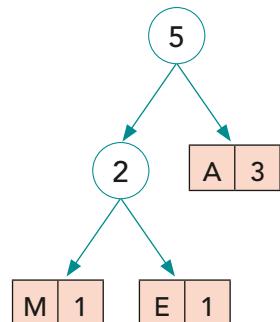
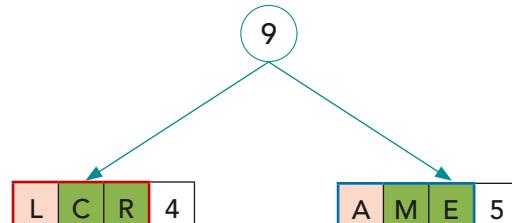


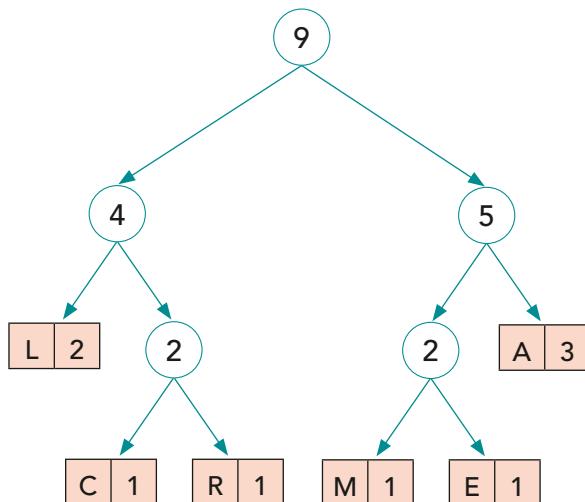
figura 6

Siamo arrivati all’ultimo passaggio, dato che sono “rimasti” solo due elementi che vengono inseriti nella struttura finale:

M-E-A	L-C-R
5	4

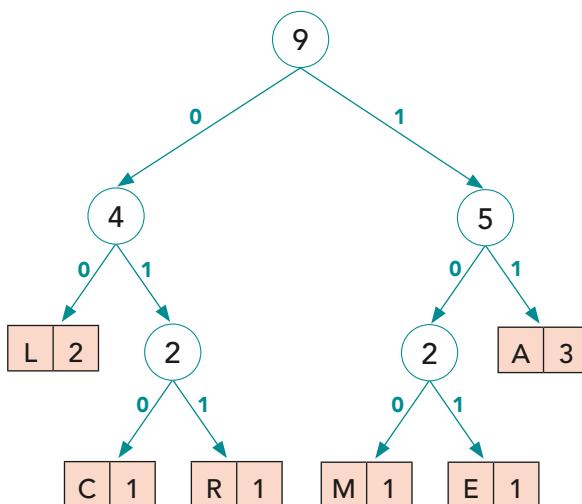


Ricostruendo ora la struttura completa otteniamo l’albero binario completo (binario perché ogni “elemento” ha due “rami”, che prendono il nome di **figlio destro** e **figlio sinistro**).



Gli elementi terminali prendono il nome di **foglie**, mentre quelli intermedi si chiamano **nodi**: al **nodo** che "sta più in alto" viene dato il nome di **radice** (in questo caso quello che ha al suo interno il valore 9). I **nodi** sono tra loro collegati da **archi** (o **rami**).

Procediamo con l'assegnazione del codice partendo dalla **radice** e inserendo uno **0** su ogni ramo del **figlio sinistro** e un **1** su ogni ramo del **figlio destro**, come nella seguente figura:



Ogni carattere avrà una codifica in bit univoca, e non ci vorrà un byte per ogni carattere: nella tabella seguente è riportata la codifica di ogni carattere.

Carattere	Codifica	Frequenza
A	11	3
L	00	2
C	010	1
R	011	1
M	100	1
E	101	1

La stringa originale viene quindi così codificata:

**CARAMELLA**

**010-11-011-11-100-101-00-00-11**

Il numero totale di bit è 22: bastano 3 byte per la stringa anziché i 9 byte iniziali, cioè 1/3.

La decodifica sarà il procedimento inverso: dato che nessuna parola codice è prefisso di un'altra, man mano che viene riconosciuta la codifica di un carattere questo viene univocamente determinato. Per la decodifica basta quindi:

- 1 individuare la prima parola codice della stringa codificata;
- 2 tradurla nel carattere originale e aggiungere tale carattere alla stringa decodificata;
- 3 rimuovere la parola codice dalla stringa codificata;
- 4 ripetere l'operazione per i successivi caratteri.

### ESEMPIO

Sapendo che la stringa

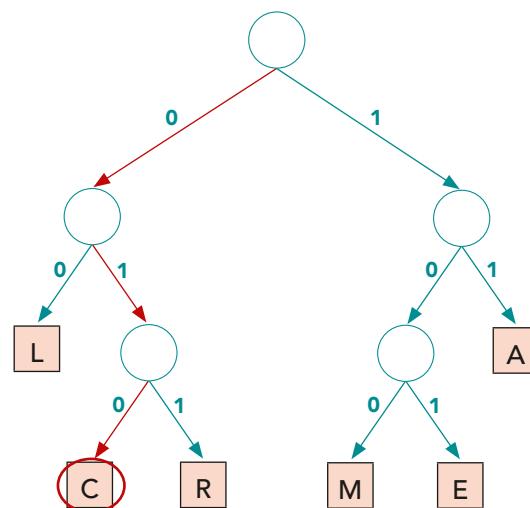
010110110011

è stata codificata con l'albero di **Huffman** dell'esempio precedente, si chiede di decodificarla.

Iniziamo ad analizzare la stringa di bit partendo dalla radice dell'albero: ►

- incontriamo uno 0 e ci muoviamo sul ramo sinistro;
- incontriamo un 1 e ci muoviamo sul ramo destro;
- incontriamo uno 0 e ci muoviamo sul ramo sinistro riconoscendo il carattere C.

Proseguiamo allo stesso modo con gli altri bit fino a che la stringa termina: è facile determinare che la parola compressa era il nome CARLA.



### Prova adesso!

• Codifica di Huffman

- 1 Una immagine bitmap ha cinque tonalità di grigio con le seguenti probabilità:

Carattere	Frequenza
r0	0.36
r1	0.20
r2	0.19
r3	0.15
r4	0.10

Effettua la codifica mediante l'algoritmo di **Huffman**.

- 2 Codifica la seguente stringa: **AABABCABDACB**

# ESERCITAZIONI DI LABORATORIO 3

## I NUMERI ROMANI CON EXCEL

Il foglio elettronico Excel permette di effettuare “direttamente” la conversione tra numeri decimali e numeri romani mediante la funzione **ROMANO()**, che ha la seguente sintassi:

**ROMANO(Num; [Forma])**

Nella sintassi della funzione ROMANO() sono previsti due argomenti:

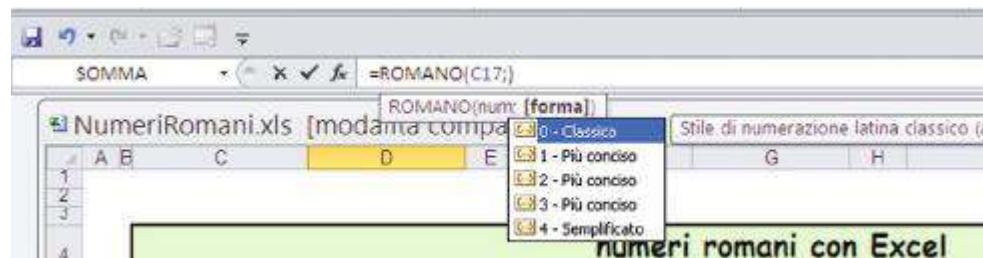
- ▶ **Num**: parametro obbligatorio che è il numero arabo che si desidera convertire;
- ▶ **Forma**: parametro facoltativo che specifica lo stile del tipo di numero romano desiderato, che può variare da **Classico** a **Semplificato** in quattro varianti: se viene omesso si attribuisce di default il valore 0, cioè classico, mentre 4 è il massimo valore che può assumere come semplificato.

### ESEMPIO

Vediamo come viene rappresentato il numero 499 nelle diverse varianti:

Arabo	Romano	Formato
499	CDXCIX	"=ROMANO(C3;0)"
499	LDVLIV	"=ROMANO(C4;1)"
499	XDIV	"=ROMANO(C5;2)"
499	VDIV	"=ROMANO(C6;3)"
499	ID	"=ROMANO(C7;4)"

Per utilizzare questa funzione è sufficiente indicare la cella dove è presente il numero arabo da convertire e selezionare eventualmente la Forma opzionale, come si può vedere nella immagine seguente:



Nella seguente immagine è riportata una esecuzione dove sono visibili i numeri nei diversi formati, da Classico (0) a Semplificato (4).



**Prova adesso!**

- Utilizzo di excel
  - Sistemi non posizionali

Costruisci un fogli elettronico come quello riportato in figura.

Una possibile soluzione è riportata nel file **NumeriRomani.xls** ed è scaricabile dalla cartella **Materiali** della sezione del sito [www.hoepliscuola.it](http://www.hoepliscuola.it) riservata a questo volume.

# ESERCITAZIONI DI LABORATORIO 4

## LA CONVERSIONE DI NUMERI IN BINARIO CON EXCEL

### ■ Premessa

Per verificare se le operazioni di conversione tra la base binaria e decimale sono esatte è sufficiente utilizzare la calcolatrice di Windows, opzione Programmatore: un'apposita sezione che permette di eseguire le conversioni tra i numeri decimali, ottali, binari ed esadecimali.



Il limite di questo programma è che permette di eseguire solo le operazioni di conversione di numeri interi e inoltre non sono presenti i convertitori per i sistemi di numerazione non multipli di 2, come ad esempio i quinari oppure i duodecimali.

Realizziamo quindi un nostro convertitore utilizzando il foglio elettronico Excel.

## ■ Conversione da decimale a binario

Come prima parte della esercitazione realizziamo un foglio elettronico che effettua la conversione da base 10 a base 2. Riportiamo sinteticamente l'algoritmo che utilizziamo:

- ➊ dividete il numero per 2 e memorizzate il resto;
- ➋ continuate a dividere per due fino a che il quoziente è zero;
- ➌ ordinate i resti nell'ordine in cui sono stati ottenuti da destra a sinistra.

Come primo passo si predispone uno schema come il seguente:

CONVERTITORE DA DECIMALE A BINARIO					
	DIVIDENDO	RESTO	POS. CIFRA	2 <sup>n</sup>	BINARIO
6	2000	0	0	0	0
7	1000	0	0	1	1
8	500	0	0	2	10
9	250	0	0	4	100
10	125	1	1	8	101
11	62	0	0	16	1010
12	31	1	1	32	1011
13	15	1	1	64	10111
14	7	1	1	128	101110
15	3	1	1	256	1011101
16	1	1	1	512	10111010
17	0	0	1	1024	101110101
18	0	0	1	0	1011101010
19	0	0	1	0	10111010100
20	0	0	1	0	101110101000
21	0	0	1	0	1011101010000
22	0	0	0	0	10111010100000
23	Prova: 2000				

Vediamo assieme la scrittura delle funzioni, in riferimento alla riga 6:

- ▶ per la colonna **resto** inseriamo la formula **[=C6-2\*C7]**
- ▶ per la colonna **2<sup>n</sup>**, che useremo come colonna di prova, e in cui scriviamo le potenze di 2 relative alla cifra binaria, dobbiamo usare l'elevamento a potenza **[=2^F6\*E6]** e la funzione **[=SOMMA(G6:G23)]** ci calcola il valore di prova sommando verticalmente le singole cifre;
- ▶ la colonna binario deve contenere il bit 0 oppure 1, solo la prima colonna contiene un valore diverso da 0, cioè se la conversione non è ancora finita: **[=SE(C6=0;"";E6)]**.

Per la rappresentazione binaria, possiamo simulare la scrittura binaria moltiplicando per 10 oppure, più elegantemente, utilizzare la funzione CONCATENA, come sotto riportato:  
**[=CONCATENA(H22;H21;H20;H19;H18;H17;H16;H15;H14;H13;H12;H11;H10;H9;H8;H7)]**



### Prova adesso!

Realizza lo schema come illustrato nella figura precedente.

Quindi utilizza le funzioni di Excel che ci devono permettere di arrotondare il quoziente all'intero inferiore e salvare il resto nel caso che esse sia frazionario.

- Convertitore decimale > binario
- Funzioni di arrotondamento e resto

Funzioni: ARROTONDA.PER.DIF  
RESTO

Realizza un nuovo foglio elettronico che utilizza queste funzioni.

## ■ Conversioni da decimale a basi diverse da 2

Nella seconda parte della esercitazione rendiamo indipendente il foglio di calcolo dalla base di destinazione della conversione, utilizzando solo riferimenti di cella:

Al posto che dividere per 2, nella colonna resto dividiamo per **\$D\$6**: in questo modo la casella può contenere qualunque base; la formula è la seguente [**=C6-\$D\$6\*C7**].

Allo stesso modo utilizzeremo la funzione  **$8^n$**  per elevare la base [**= \$D\$6^F6\*E6**].

Il foglio risulta essere il seguente:

Ora possiamo calcolare la conversione di un numero in tutte la altre basi, come 3, 5, 8 ecc. Osservate che l'algoritmo “funziona” anche con la base 10 (il numero viene riscritto uguale).

Possiamo anche mettere parametrica l’etichetta della colonna g in modo che il titolo venga automaticamente ripreso dalla cella D8 con la formula [=CONCATENA(D6,"^n")]

## ■ Conversione da decimale a esadecimale

Sappiamo che la notazione binaria è molto ingombrante e poco sintetica, e nei calcolatori viene utilizzata la rappresentazione in base 16.

In base 16 abbiamo bisogno di 16 cifre, ma noi ne conosciamo solo 10, per questo, dopo il 9, si usano le lettere; le 16 cifre sono: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Con un byte possiamo denotare 1 Byte = 256 valori =  $16^2$ ; quindi 1 Byte viene descritto da 2 cifre in base 16: 00, 01, ... FE, FF.

L’algoritmo di conversione utilizzato sino a ora va bene per il calcolo dei resti: dobbiamo solo aggiungere un ulteriore controllo che effettua la conversione dei valori tra 10 e 15 in lettera.

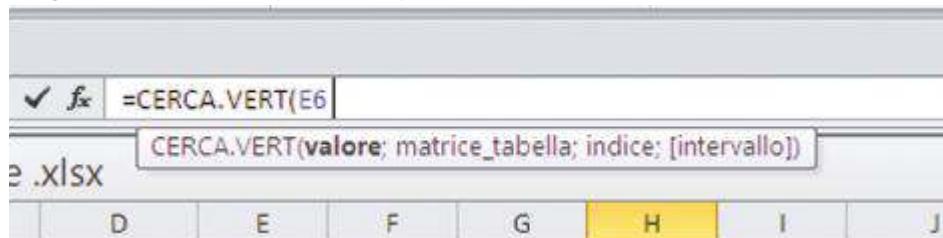
A tal fine aggiungiamo una tabella di conversione e la scorriamo analizzando i resti presenti nella colonna E per trasformarli nella cifra esadecimale nella colonna H.

CONVERTITORE DA DECIMALE A ESADECIMALE										
Decimale	Divisore	Resto	posta cifra	prova	trasforma 16-16 A-F	Esadecimale	Tabella di conversione			
14255	16	15	0	15	F	F	0	0	1	1
		10	1	160	A	A	2	2	3	3
890			2	1792	7	7	4	4	5	5
55		7	3	12288	3	3	6	6	7	7
3		0	4	0	0	0	8	8	9	9
0		0	5	0	0	0	10	A	11	B
0		0	6	0	0	0	12	C	13	D
0		0	7	0	0	0	14	E	15	F
0		0	8	0	0	0				
0		0	9	0	0	0				
0		0	10	0	0	0				
0		0	11	0	0	0				
0		0	12	0	0	0				
prova: 14255										

Nelle celle della colonna H inseriamo la seguente formula:

[=CERCA.VERT(E6;M\$6:N\$21;2)]

che ha la seguente struttura (prototipo):



Dove:

- valore: valore da ricercare;
- matrice\_tabella: coordinate “spigoli estremi” della tabella di conversione;
- indice: colonna dalla quale prendere il valore di ritorno.



**Prova adesso!**

- Convertitore decimale > esadecimale
- Convertitore decimale > vigesimali

Realizza lo schema come illustrato.

Quindi apporta le modifiche necessarie per effettuare la conversione tra la base decimale e quella vigesimale, dove si utilizzano ad esempio le seguenti cifre:

10	11	12	13	14	15	16	17	18	19
A	B	C	D	E	F	G	H	M	N

**ESERCITAZIONI DI LABORATORIO 5**

# CONVERSOINE ALLA BASE DECIMALE

## ■ Convertitore dalle diverse basi a decimale con 3 cifre decimali

Si vuole realizzare un convertitore con 5 cifre intere e 3 cifre decimali che permetta di passare dalle diverse basi a quella decimale.

## Da binario a decimale

Mandiamo in esecuzione **Excel** (o un qualunque foglio di calcolo) e iniziamo la “costruzione” di un convertitore da **binario** a **decimale** predisponendo una riga dove scriviamo in ogni cella il valore dell’esponente partendo dal valore maggiore (5) fino ad arrivare alla terza cifra decimale (-3).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1														
2														
3														
4														
5	esponente	5	4	3	2	1	0	-1	-2	-3				

Predisponiamo le celle nelle quali successivamente collocheremo le formule per fare eseguire al foglio di calcolo le diverse operazioni :

Nella cella O9 faremo calcolare il risultato, cioè il numero decimale ottenuto dalla conversione dei singoli bit che inseriremo nelle celle E8... M8.

	formula 1
--	-----------

Nella **riga 9** eseguiamo il calcolo di ogni singolo bit, sapendo che per il termine **n-simo** vale la seguente formula:

$$\text{bit}_n = x * 2^n \text{ con } x \text{ che può valere 0 oppure 1}$$

Possiamo quindi scrivere la formula che ci permette di calcolare il valore delle singole celle partendo dalla cella E9:

	formula 2
--	-----------

dove

**E8**: è la cella dove inseriremo il valore del bit, cioè 0 o 1;

**\$D9**: è la cella dove inseriremo il valore della base, in questo caso 2;

**E\$5**: è la cella dove abbiamo inserito l'esponente, in questo caso 5.

Copiamo la formula in tutte le celle della riga 9 e inseriamo il numero binario **110101** ottenendo il seguente risultato:

	BINARIO	(inserire 0/1)	parte intera	parte decimale	decimale
BASE	2	32 16 0 4 0 1	0,000 0,000 0,000	= 53,000	

Inseriamo ora un valore con parte decimale, come il numero **1000.010** e otterremo:

	BINARIO	(inserire 0/1)	parte intera	parte decimale	decimale
BASE	2	0 0 1 0 0 0 0 1 0	0,000 0,250 0,000	= 8,250	

## ■ Conversione da quinario a decimale

Predisponiamo ora un secondo gruppo di righe per effettuare la conversione da quinario a decimale: nella cella D13 inseriamo il valore della base 5.

QUINARIO	(inserire 0/1)	0 0 0 0 0 0 0 0 0	decimale
BASE	5	0 0 0 0 0 0,000 0,000 0,000	= 0,000

Nella cella O13 faremo calcolare il risultato, cioè il numero decimale ottenuto dalla conversione dei singoli bit che inseriremo nelle celle E13... M13 (**formula 1**).

Nella riga 13 eseguiamo il calcolo di ogni singolo bit, sapendo che per il termine **n-simo** con la medesima **formula 2** utilizzata per la base precedente, in quanto la base viene letta come parametro dalla cella D13: la ricopiamo quindi in ogni cella.

Inseriamo il numero quinario 123 ottenendo il seguente risultato:

11		QUINARIO	(inserire 0/4)	0	0	0	1	2	3	0	0	0	= decimalle
12		BASE	5	0	0	0	25	10	3	0,000	0,000	0,000	= 38,000
13													
14													

Effettuiamo ora una verifica del corretto funzionamento anche per la parte decimale inserendo 1000.123:

11		QUINARIO	(inserire 0/4)	0	0	1	0	0	0	1	2	3	= decimalle
12		BASE	5	0	0	125	0	0	0	0,200	0,080	0,024	= 125,304
13													
14													

## ■ Conversione da ottale a decimale

Analogo discorso viene fatto per la conversione tra base **ottale** e **decimale**

10		OTTALE	(inserire 0/7)	0	0	0	0	0	0	0	0	0	= decimalle
11		BASE	8	0	0	0	0	0	0	0,000	0,000	0,000	= 0,000
12													
13													
14													

## ■ Conversione da duodecimale a decimale

Il sistema **duodecimale** non viene utilizzato in informatica ma in Inghilterra, fino a qualche tempo fa, si usava per le monete un sistema a base 12.

Oggi, comunque, le uova vengono ancora vendute a dozzine e 12 sono le ore del giorno!

Realizziamo quindi un convertitore per passare dalla base **dodici** a **decimale**

19		DUODECIMALE	(inserire 0/11)	0	0	0	0	0	0	0	0	0	= decimalle
20		BASE	12	0	0	0	0	0	0	0,000	0,000	0,000	= 0,000
21													
22													

Nella successiva figura viene riportato un esempio completo dove ogni cifra è stata posta al valore 1 per poter confrontare le differenze dei diversi pesi:

CONVERTITORE DALLE DIVERSE BASI A DECIMALE																
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2																
3																
4																
5					esponente	5	4	3	2	1	0	-1	-2	-3		
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																
23																



*Prova adesso!*

- Realizzazione di un convertitore
  - Conversione numero dalle diverse basi

Dopo aver realizzato un foglio di calcolo simile a quello di figura esegui manualmente le seguenti conversioni e successivamente verificale con il convertitore da te realizzato.

### Da binario a decimale

10010.1; 10101.011; 11001.100; 10111.111; 1000.001.

### Da quinario a decimale

10020.1: 10203.041: 11021.103: 10314.141: 1004.004

### Da ottale a decimale

10026 12: 10305 017: 110031 152: 20406 121: 2007 007

Da duodecimale a decimale ( $A=10$ ,  $B=11$ )

10020.4: 10709.011: 1200A.102: 10319.1A1: 10305.709.

## Convertitore da esadecimale a decimale

La codifica in esadecimale contiene anche delle lettere e quindi per poter effettuare in **Excel** la conversione tra esadecimale e decimale è necessario innanzi tutto predisporre una tabella di conversione tra i sedici simboli della base esadecimale e i corrispondenti valori decimali, come quella riportata di seguito:

Predisponiamo ora un insieme di 4 celle dove inseriremo il numero esadecimale (**E11**...  
**H11**) e la cella in cui andremo a calcolare il risultato della conversione **J11**.

Prima di poter moltiplicare ogni cifra per il corrispondente peso legato alla posizione è necessario effettuare una conversione tra le lettere A.B.C.D.E.F e il loro valore numerico mediante la tabella che abbiamo predisposto nelle righe 6 e 7: ci avvarremo per ogni cifra della funzione

A	B	C	D	E	F	G
1						

che prende il valore presente nella cella H11 e cerca il valore presente nella prima riga della tabella di coordinate C6 e R7 e ritorna il valore della riga indicata (2).

Ricopiamo questa formula in tutte e quattro le celle e successivamente nelle celle della riga 14, a partire dalla E14, effettuiamo la moltiplicazione per il valore del peso.

A	B	C	D	E	F
H14					

e nella cella G6 inseriamo la somma dei parziali così ottenuti:

J11	f <sub>x</sub>	=SOMMA(E14:H14)



### Prova adesso!

- Realizzazione di un convertitore con tabella di conversione

Verifichiamo il funzionamento inserendo il numero 1A7F<sub>H</sub> e ottenendo:

Esadecimale				Decimale
	1	A	7	F
peso	4096	256	16	1
corrispondenza	1	10	7	15
valore	4096	2560	112	15

Dopo aver realizzato un foglio di calcolo simile a quello di figura esegui manualmente le seguenti conversioni e successivamente verificale con il convertitore da te realizzato:

123A; 2468; 147D; 222E; 357E.

Completa il foglio aggiungendo anche le cifre decimali e verificane il funzionamento con i seguenti numeri :

1.123; 2.7AB; 3.F71; 4.EEE; 5.B2C.

Infine modifica il convertitore duodecimale inserendo anche per esso una tabella di corrispondenza per le coppie A-10 e B-11.

# ESERCITAZIONI DI LABORATORIO 6

## CONVERSIONE TRA LE BASI BINARIE

### ■ Premessa

Nella conversione tra numeri che hanno base di partenza e/o di arrivo che sono tra loro potenze possiamo utilizzare delle procedure di conversione semplificate.

#### Base di partenza potenza della base di arrivo

Quando tra la base B1 e la base B2 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.

#### Base di arrivo potenza della base di partenza

Quando tra la base B1 e la base B2 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.

In questa esercitazione realizzeremo dei semplici programmi per effettuare le conversioni utilizzando un foglio elettronico.

Dalla versione 2007 sono presenti in Excel alcune funzioni che eseguono automaticamente la conversione, come le seguenti

=OCT.HEX(<numero>)	=OCT.BINARIO(<numero>)
=HEX.BINARIO(<numero>)	=HEX.OCT(<numero>)
=BINARIO.HEX(<numero>)	=BINARIO.OCT(<numero>)

Naturalmente NON possono essere utilizzate per la realizzazione dei nostri programmi (sarebbe troppo facile...).

## ■ Base ottale

Riportiamo la tabella di conversione tra cifre binarie e sistema di numerazione ottale: ogni **3 cifre** binarie corrispondono a un numero ottale (e viceversa) dato che  $8 = 2^3$ .

binario	ottale
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

## Da binario a ottale

Effettuiamo la conversione in ottale del numero binario **10000000101** (corrispondente al decimale 1029): raggruppiamo le cifre 3 a 3, partendo dalla posizione meno significativa, ovvero quella più a destra:

binario	binario raggruppato	ottale raggruppato	ottale
10000000101	10 000 000 101	2 0 0 5	2005

Abbiamo ottenuto le quattro cifre 2|0|0|5 corrispondenti al numero ottale 2005.

Effettuiamo la conversione in ottale del numero binario **1101001101** (corrispondente a  $845_{10}$ ):

binario	binario raggruppato	ottale raggruppato	ottale
1101001101	1 101 001 101	1 5 1 5	1515



**Prova adesso!**

• Convertitore in Excel

Realizza ora un semplice convertitore in Excel da **binario a ottale**.

Per ogni singola cifra ottale devi sommare 3 cifre binarie:

- il bit **più significativo** viene moltiplicato per  $2^2$ ;
- il bit **centrale** lo viene moltiplicato per  $2^1$ ;
- il bit **meno significativo** lo si somma direttamente.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																

Conversione da binario a ottale															
<b>Binario</b>															
0 0 1 1 0 1 0 0 1 1 0 1															
<b>Ottale</b>															
1 5 1 5															

## Da ottale a binario

La conversione da ottale a binario viene effettuata con il metodo inverso: ad esempio, il numero ottale 7246 (corrispondente al decimale 3750) lo scomponiamo nelle singole cifre e ciascuna di esse la convertiamo nei tre bit che lo codificano:

ottale	ottale raggruppato	binario raggruppato	binario
7246	7 2 4 6	111 010 100 110	111010100110



### Prova adesso!

• Convertitore in Excel

Realizza ora un semplice convertitore in Excel da **ottale** a **binario**.

Per ogni singola cifra ottale sappiamo che si devono ottenere 3 cifre binarie:

- il bit **più significativo** è banalmente il valore della divisione per  $2^2$ ;
- il bit **centrale** lo si ottiene prendendo "quello che resta" del numero iniziale (cioè il numero iniziale meno quanto è già espresso dal bit più significativo) e dividendolo per  $2^1$ ;
- il bit **meno significativo** lo si ottiene prendendo "quello che resta" del numero iniziale e dividendolo per  $2^0$  (cioè lo si lascia così com'è).

Supponiamo che le singole cifre ottali siano tutte in caselle diverse, diciamo A2,B2,C2... E suppongo anche che le singole cifre binarie del risultato possano comparire tutte in caselle diverse, diciamo A1,B1,C1 (i tre bit che convertono A2); D1,E1,F1 (i tre bit che convertono B2);...

Le funzioni che devi mettere in A1,B1, ... sono:

```
A5=INT(D7/4)
B5=INT((D7-A5*4)/2)
C5=INT(D7-A5*4-B5*2)

D5=INT(E7/4)
E5=INT((E7-D5*4)/2)
F5=INT(E7-D5*4-E5*2)

...
```

Testiamo il programma convertendo il numero 7246<sub>8</sub> come riportato nella seguente figura.

Conversione da ottale a binario	
Binario	1 1 1 0 1 0 1 0 0 1 1 0
Ottale	7 2 4 6

## ■ Esadecimale

Riportiamo la tabella di conversione tra cifre binarie e sistema di numerazione esadecimale:  
**4 cifre** binarie corrispondono a un numero esadecimale (e viceversa) dato che  $16 = 2^4$ .

binario	esadecimale	binario	esadecimale
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

### Da binario a esadecimale

Effettuiamo la conversione in esadecimale del numero binario **10000000101** (corrispondente al decimale 1029): raggruppiamo le cifre 4 a 4, partendo dalla posizione meno significativa, ovvero quella più a destra:

binario	binario raggruppato	HEX raggruppato	esadecimale
10000000101	100 0000 0101	4 0 5	405

Effettuiamo la conversione in HEX del numero binario **1101001101** (corrispondente a  $845_{10}$ ).

binario	binario raggruppato	HEX raggruppato	esadecimale
10000000101	11 0100 1101	3 4 D	34D



### Prova adesso!

• Convertitore in Excel con tabella

Realizza ora un semplice convertitore in Excel da binario a esadecimale come quello indicato in figura introducendo una tabella conversione per trasformare  
 $A \rightarrow 10, B \rightarrow 11, C \rightarrow 12, D \rightarrow 13, E \rightarrow 14, F \rightarrow 15$ .

Conversione da binario a esadecimale	
Binario	1 0 1 1 0 0 1 1 1 1 1 1
Esadecimale	B 3 F 11 3 15

## Da esadecimale a binario

La conversione da esadecimale a binario viene effettuata con il metodo inverso: ad esempio, il numero esadecimale 1E22A6 (corrispondente al decimale 1974950) lo scomponiamo nelle singole cifre e ciascuna di esse la convertiamo nei quattro bit che lo codificano:

esadecimale	HEX raggruppato	binario raggruppato	binario
1E22A6	1 E 2 2 A 6	1 1110 0010 0010 1010 0110	111100010001010100110



### Prova adesso!

• Convertitore in Excel con tabella

Realizza ora un convertitore da binario a esadecimale come quello indicato in figura introducendo una tabella di conversione per trasformare:

A → 10, B → 11, C → 12, D → 13, E → 14, F → 15.

Per ogni singola cifra ottale sappiamo che si devono ottenere 4 cifre binarie:

- il **bit più significativo** è banalmente il valore della divisione per  $2^3$ ;
- il **secondo** bit lo si ottiene prendendo "quello che resta" del numero iniziale (cioè il numero iniziale meno quanto è già espresso dal bit più significativo) e dividendolo per  $2^2$ ;
- il **terzo** bit lo si ottiene prendendo "quello che resta" del numero iniziale sottraendo i primi due valori ottenuti e dividendolo per  $2^1$ ;
- il **bit meno significativo** lo si ottiene prendendo "quello che resta" del numero iniziale e dividendolo per  $2^0$  (cioè lo si lascia così com'è).

Supponiamo che le singole cifre esadecimali siano tutte in caselle diverse, diciamo D7, E7, F7 e supponiamo anche che le singole cifre binarie del risultato si possano inserire in caselle diverse, diciamo D5, E5, F5, G5 (i quattro bit che convertono D7) ecc.

```
D5=INT(D8/8)
E5=INT((D8-D5*8)/4)
F5=INT((D8-D5*8-E5*4)/2)
G5=INT(D8-D5*8-E5*4-F5*2)
```

### Conversione da esadecimale a binario

Esadecimale

A	7	A
15	7	10

Binario

1 1 1 1 0 1 1 1 1 0 1 0

## ■ Conversione tra ottale ed esadecimale

Dagli esempi precedenti possiamo notare come si possa facilmente passare da qualunque rappresentazione avente come base potenze di 2 alla rappresentazione binaria e viceversa; perciò il passaggio da rappresentazione ottale a rappresentazione esadecimale si farà con un primo passaggio dalla rappresentazione ottale alla rappresentazione binaria e un successivo dalla rappresentazione binaria alla rappresentazione esadecimale:

<b>ottale</b>	<b>ottale raggruppato</b>	<b>binario raggruppato</b>	<b>binario</b>
7421246	7 4 2 1 2 4 6	111 100 010 001 010 100 110	111100010001010100110

<b>binario</b>	<b>binario raggruppato</b>	<b>HEX raggruppato</b>	<b>esadecimale</b>
111100010001010100110	1 1 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0	1 E 2 2 A 6	1E22A6

Si noti ancora come il numero di cifre binarie da raggruppare sia l'esponente di 2 corrispondente alla base.



### Prova adesso!

- Convertitore in Excel con tabella

Realizza ora un convertitore da **ottale** a **esadecimale** (e viceversa) come quello indicato in figura introducendo una tabella di conversione per trasformare

A → 10, B → 11, C → 12, D → 13, E → 14, F → 15.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2																	
3	<b>Conversione da ottale a esadecimale</b>												<b>Tabella di conversione</b>				
4													0	0			
5													1	1			
6													2	2			
7	<b>Ottale</b>		1 3 4										3	3			
8													4	4			
9	<b>Binario</b>		0 0 1			0 1 1	1 0 0						5	5			
10													6	6			
11	<b>Binario</b>		0 0 1			0 1 1	1 1 0	0 0					7	7			
12													8	8			
13	<b>Esadecimale</b>		0 D 9										9	9			
14													10	A			
15													11	B			
16													12	C			
17													13	D			
18													14	E			
19													15	F			
20																	
21																	

# 2

## I codici digitali

- L1 **Codici digitali pesati**
- L2 **Codici digitali non pesati**
- L3 **La correzione degli errori**

### Esercitazioni di laboratorio

- 1 Simulare una trasmissione seriale con Excel; 2 Calcolo del check digit nei codici EAN 8 ed EAN 13; 3 Un programma per la codifica di Hamming

#### Conoscenze

- Comprendere le differenze tra codifica a lunghezza fissa e variabile
- 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

#### Competenze

- Conoscere il codice ASCII e Unicode
- Codificare e decodificare numeri e codici
- Codificare in codice BCD, eccesso 3 e Gray
- Codificare a sette segmenti e a matrice di punti
- Codificare e decodificare con QR Code
- Saper codificare con i codici di Hamming

#### Abilità

- Convertire numeri e codici rappresentati secondo sistemi diversi
- Eseguire somma e sottrazione in BCD
- Correggere l'errore con byte di checksum
- Individuare l'errore con il codice di parità
- Correggere l'errore con il codice di Hamming

### AREA *digitale*

-  ▶ Esercizi
-  ▶ Tabella caratteri ASCII standard ed esteso
  - ▶ Confronto codifica dei numeri da 0 a 9 nei diversi codici
  - ▶ Come generare e leggere i QR Code
  - ▶ Errori e probabilità
  - ▶ Combinazioni a distanza

# Codici digitali pesati

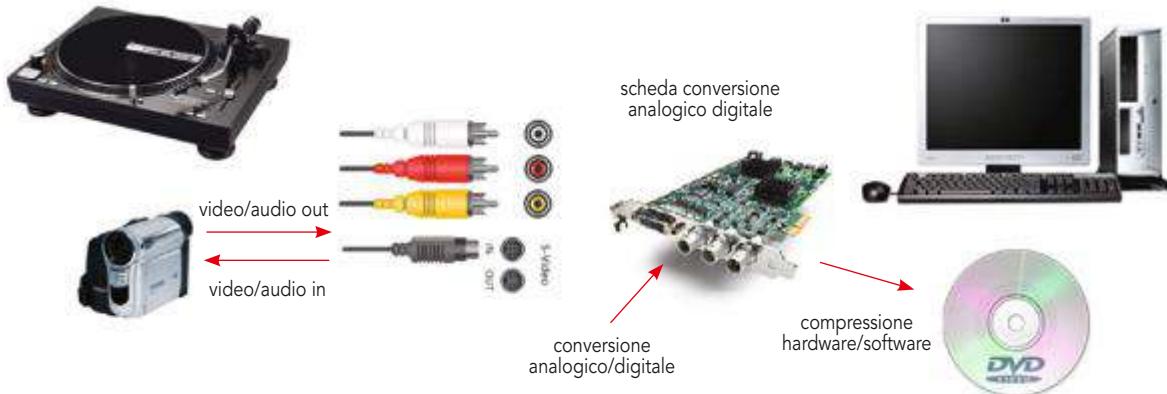
In questa lezione impareremo...

- 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 alla codifica dell'informazione

Un **calcolatore elettronico** elabora esclusivamente numeri sotto forma di valori digitali che devono essere trasformati in sequenze di valori segnali a due valori (binari).

I sistemi di elaborazione devono tuttavia scambiare informazioni con il mondo esterno in ingresso e in uscita: questi in generale sono segnali analogici che devono essere trasformati in digitale (e viceversa) da appositi convertitori **analogici/digitali (A/D)** e **digitali/analogici (D/A)** a seconda della natura del segnale (testo, numero, suono, video ecc.).



Anche le semplici informazioni che l'uomo deve scambiare con la macchina devono essere comprensibili e quindi “possibilmente” visualizzate sotto forma di caratteri alfanumerici (numeri decimali, lettere dell'alfabeto, simboli di punteggiatura, simboli matematici).

Dalla parte della macchina, invece, i segnali digitali vengono codificati in modo tale che possano essere elaborati e memorizzati da un computer: i calcolatori lavorano soltanto con i numeri (in particolare solo con 0 e 1), ma devono poter trattare anche altre entità, per questa ragione sono stati inventati i **codici**.



### CODICE, ALFABETO E STRINGA

Si dice **codice** una regola che consente di rappresentare un dato insieme di simboli mediante un altro insieme di simboli.

Si definisce **alfabeto** (e si indica con  $\Sigma$ ) l'insieme finito di **simboli** (caratteri) distinti.

Si definisce **parola** o **stringa** una sequenza finita di simboli (anche ripetuti) di quello stesso **alfabeto**.

### ESEMPIO

Alcuni **alfabeti** sono i seguenti:

- **unario U** = {1}
- **binario B** = {0, 1}
- **inglese I** = {A, B, C, ..., Z}
- **alfanumerico AB** = {A, B, ..., Z, 0, 1, ..., 9}
- **caratteri giapponesi** = { ... }
- **caratteri arabi** = { ... }

Le seguenti **parole** possono essere originate da **alfabeti** differenti:

- **1111001101** è una parola di **B** o di **AB**
- **AB120C** è una parola di **AB**
- **111111** è una parola di **B** o di **U**

Il linguaggio umano è molto lontano dal linguaggio del calcolatore: è necessario superare l'incomunicabilità fra i due linguaggi mediante l'adozione di opportune convenzioni (**codifiche**), mediante le quali è possibile rappresentare in modo univoco un certo numero di simboli con configurazioni di bit prestabilite.



### SISTEMA DI CODIFICA

Dato un alfabeto A, composto di  $a$  simboli, un **sistema di codifica** (o semplicemente detta **codifica**) è un metodo (procedimento) per rappresentare gli  $a$  simboli di A tramite  $b$  simboli di un altro alfabeto B stabilendo una corrispondenza biunivoca tra gli stessi.

### ESEMPIO

Sia

- |                        |                                  |
|------------------------|----------------------------------|
| $A = \{ R; V; G; B \}$ | un alfabeto di quattro caratteri |
| $B = \{ 0, 1 \}$       | un alfabeto di due caratteri     |

Si vuole codificare la seguente stringa di A in B:

B R R V B

Stabilisco una semplice corrispondenza tra i simboli dell'alfabeto a e la loro posizione che

codifico in numero binario, quindi:

$$\begin{aligned} R_A &= 0_{10} = 00_B \\ V_A &= 1_{10} = 01_B \\ G_A &= 2_{10} = 10_B \\ B_A &= 3_{10} = 11_B \end{aligned}$$

L'insieme dei simboli originari da codificare si chiama **insieme sorgente** (o esterno), l'insieme dei simboli codificati si chiama **insieme codificato** (o interno).



## 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

### Lunghezza del codice e codice ridondante

Sappiamo che il computer elabora solo **segnali binari** (0 e 1) e quindi tutti dati digitali devono essere codificati in **codice binario**, in modo tale da quale mantenere una corrispondenza a biunivoca tra i simboli da codificare (simboli originari)



### CODICE BINARIO

Un **codice binario** è la codifica dei simboli di un **alfabeto  $\Sigma$**  mediante stringhe di **bit**.

Se **C** è la **cardinalità** (numero di elementi, cioè parole diverse) di  $\Sigma$ , il numero **n** di bit da utilizzare per codificare tutti i simboli di  $\Sigma$  con un codice binario deve essere tale che:

$$C \leq 2^n$$

ossia:

$$n \geq \log_2(C) = M$$

$M$  è il minimo numero di bit necessario per codificare  $\Sigma$ :

- se  $n = M$  il codice è non ridondante (usa il minimo numero di bit per codificare tutti i simboli di  $\Sigma$ );
- se  $n > M$  il codice è ridondante (usa più bit del necessario per codificare tutti i simboli di  $\Sigma$ ).

### ESEMPIO

Alfabeto = {A,B,C,...,Z}

C = 26 cardinalità

$n \geq \log_2 26$  numero di bit

$n \geq 5$

$M = 5$  minimo numero di bit

Alfabeto = {0,1,2,...,9}

C = 10 cardinalità

$n \geq \log_2 10$  numero di bit

$n \geq 4$

$M = 4$  minimo numero di bit



### 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.

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.

Inoltre vedremo come è possibile sfruttare la ridondanza per aggiungere informazioni utili alla trasmissione del dato e per facilitare la sua interpretazione e correzione nel caso che si rovini, cioè che qualche bit si modifichi cambiando il suo valore, passando cioè da 0 a 1 oppure da 1 a 0 in modo indesiderato.

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.

In questa lezione verrà effettuata una carrellata sui principali **codici digitali pesati** a lunghezza fissa.

## ■ 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 semi-grafici e altri simboli di uso meno comune.

Vediamo di seguito alcuni esempi di codifica.

01001000	01100101	01101100	01101100	01101111	00101110
H	e	I	I	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.



### Prova adesso!

• Codice ASCII esteso



#### PRENDI CARTA E PENNA

Codifica le seguenti scritte (compresa la cornice) in codice **ASCII** esteso:

<paolo>	<@hoepli.it>
---------	--------------

Decodifica le seguenti parole dal codice **ASCII**:

01100011 01101001 01000000 01101111 01011111 01101101 01000000 01110010 01100101

01101001 01101110 01100110 01101111 01110010 01101101 01100001 01110100 01101001  
01100011 01100001

01010111 00100000 01101100 01100001 00100000 01110011 01110001 01110101 01101111  
01101100 01100001

### AREA digitale



Tabella caratteri ASCII standard ed esteso

## 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.

Tutte le informazioni in merito al codice **UNICODE** sono reperibili nel sito specifico a esso dedicato all'indirizzo <http://www.unicode.org/>

## ■ 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).

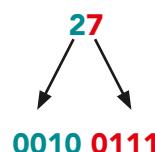
Mediante questa codifica utilizziamo 4 bit per rappresentare le cifre decimali che vanno dallo 0 al 9 in codice binario ( $2^4 = 16$ , quindi abbiamo un codice ridondante in quanto 6 configurazioni sono inutilizzate).

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			
	8	4	2	1
cifra/peso				
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:

$$\begin{aligned} 27_{10} &= 11011_2 = 0010\ 0111_{BCD} \\ 127_{10} &= 1111111_2 = 0001\ 0010\ 0111_{BCD} \\ 379_{10} &= 101111011_2 = 0011\ 0111\ 1001_{BCD} \end{aligned}$$

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,  $(0101)_{BCD} = 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = (5)_{10}$ .

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:

$$\begin{array}{r} 5 + 0101 + \\ 8 = 1000 = \\ \hline 13 \quad 1101 \end{array}$$

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:

$$\begin{array}{r}
 1101 + \\
 0110 = \\
 \hline
 0001\ 0011
 \end{array}
 \quad \begin{array}{l}
 \text{primo risultato} \\
 \text{numero 6}
 \end{array}$$

### ESEMPIO

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

$$\begin{array}{r}
 25 + \\
 5 = \\
 \hline
 30
 \end{array}
 \quad \begin{array}{r}
 0010\ 0101 + \\
 0101 = \\
 \hline
 0010\ 1010 +
 \end{array}
 \quad \begin{array}{l}
 1010 \text{ non ammesso} \\
 0110 = \text{ sommo il numero 6} \\
 \hline
 0011\ 0000
 \end{array}$$

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:

$$\begin{array}{r}
 33 - \\
 4 = \\
 \hline
 29
 \end{array}
 \quad \begin{array}{r}
 0011\ 0011 - \\
 0100 = \\
 \hline
 0010\ 1111 -
 \end{array}
 \quad \begin{array}{l}
 1111 \text{ non ammesso} \\
 0110 = \text{sottraggo il numero 6} \\
 \hline
 0010\ 1001 \quad \text{ottengo 29}
 \end{array}$$

## 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 quattro 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  $9 - 3 = 6$ :

$$\begin{aligned} C_9[3] &= C_9[0011] = 1100 = (6)_{10} \\ C_9[7] &= C_9[1101] = 0010 = (2)_{10} \end{aligned}$$

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: ►

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. ►

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

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

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.

Dec.	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

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: ►

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.

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

### ■ 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.

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.

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

## AREA digitale



Confronto sulla codifica dei numeri da 0 a 9 nei diversi codici

## Verifichiamo le conoscenze

### 1. Risposta multipla

**1** Quale tra i seguenti codici ha lunghezza fissa?

- a. il codice fiscale
- b. il codice di avviamento postale
- c. il codice di matricola
- d. la partita IVA
- e. il numero di telefono
- f. il codice sanitario

**2** Il codice ASCII:

- a. ha lunghezza 7 bit
- b. ha lunghezza 8 bit
- c. ha lunghezza 16 bit
- d. ha lunghezza dipendente dalle applicazioni

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

- a. è compatibile con il codice ASCII
- b. ha lunghezza 16 byte
- c. è utilizzato nel Web
- d. non è stato ancora completamente definito

**4** A quale numero corrisponde la codifica BCD 00011000?

- |       |                           |
|-------|---------------------------|
| a. 9  | d. 24                     |
| b. 11 | e. nessuno dei precedenti |
| c. 18 |                           |

**5** A quale numero corrisponde la codifica Aiken 00101100?

- |       |                           |
|-------|---------------------------|
| a. 26 | d. 44                     |
| b. 28 | e. nessuno dei precedenti |
| c. 32 |                           |

**6** A quale numero corrisponde la codifica quinario 01110001?

- |       |                           |
|-------|---------------------------|
| a. 26 | d. 44                     |
| b. 28 | e. nessuno dei precedenti |
| c. 32 |                           |

**7** A quale numero corrisponde la codifica biquinario 010001?

- |       |                           |
|-------|---------------------------|
| a. 2  | d. 17                     |
| b. 5  | e. nessuno dei precedenti |
| c. 11 |                           |

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

- |      |                           |
|------|---------------------------|
| a. 2 | d. 7                      |
| b. 5 | e. nessuno dei precedenti |
| c. 6 |                           |

### 2. Vero o 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

### 1. Esercizi

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	Quinario	2/5	Biquinario
5					
12					
21					
38					
69					

3 Esegui le seguenti operazioni BCD.

$$47 + \underline{\quad} \quad \underline{\quad} +$$

$$78 - \underline{\quad} \quad \underline{\quad} -$$

$$12 = \underline{\quad} \quad \underline{\quad} =$$

$$24 = \underline{\quad} \quad \underline{\quad} =$$

$$\underline{\quad} - \underline{\quad}$$

$$\underline{\quad} - \underline{\quad}$$

$$59 \quad \underline{\quad} \quad \underline{\quad}$$

$$14 \quad \underline{\quad} \quad \underline{\quad} -$$

$$47 + \underline{\quad} \quad \underline{\quad} +$$

$$22 - \underline{\quad} \quad \underline{\quad} -$$

$$23 = \underline{\quad} \quad \underline{\quad} =$$

$$4 = \underline{\quad} \quad \underline{\quad} =$$

$$\underline{\quad} - \underline{\quad}$$

$$\underline{\quad} - \underline{\quad}$$

$$70 \quad \underline{\quad} \quad \underline{\quad}$$

$$19 \quad \underline{\quad} \quad \underline{\quad} -$$

sommo

..... = sottraggo

$$\underline{\quad} - \underline{\quad}$$

$$\underline{\quad} - \underline{\quad}$$

$$70 \quad \underline{\quad} \quad \underline{\quad}$$

$$\underline{\quad} - \underline{\quad}$$

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



# Codici digitali non pesati

In questa lezione 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: ► 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).

Numero decimale	Numero eccesso 3			
	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** 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 Eseguiamo la somma di 2 + 5

$$\begin{array}{r} 2 + 0101 + \\ 5 = 1000 = \\ \hline 7 \quad 1101 \end{array}$$

risultato errato

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

$$\begin{array}{r} 1101 - \quad (\text{primo risultato}) \\ 0011 = \quad (\text{numero 3 in binario}) \\ \hline 1010 \quad \text{numero 7: risultato corretto} \end{array}$$

### ESEMPIO Eseguiamo la somma di 2 + 8

$$\begin{array}{r} 2 + 0101 + \\ 8 - 1011 = \\ \hline xx \quad 10000 \end{array}$$

risultato errato dato che supera il 9

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

$$\begin{array}{r} 1 \ 0000 + \quad (\text{primo risultato}) \\ 0011 \ 0011 = \quad (\text{numero 3 in binario}) \\ \hline 0100 \ 0011 \quad \text{numeri 1 e 0 rappresentati in eccesso 3} \end{array}$$

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

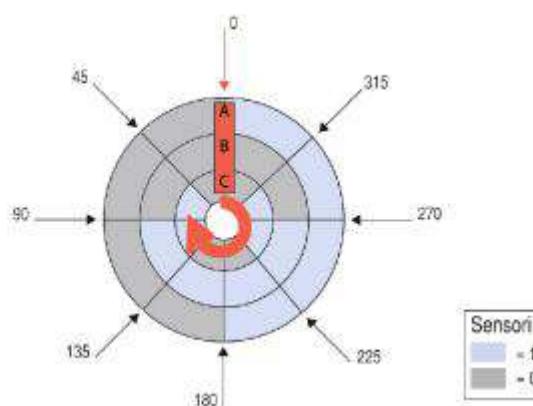
Il termine **encoder** è utilizzato in elettronica per indicare un dispositivo elettromeccanico che esegue la funzione di trasduttore di posizione angolare: esso converte la posizione angolare del suo asse rotante in numerici digitali che indicano il valore dell'angolo (chiamato **resolver**).

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	011
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 0 0	0 0 0 0
1	0 1	0 0 1	0 0 0 1
		0 1 1	0 0 1 1
	1 1	0 1 0	0 0 1 0
1 0			0 1 1 0
		1 1 0	0 1 1 1
		1 1 1	0 1 0 1
		1 0 1	0 1 0 0
		1 0 0	
			1 1 0 0
			1 1 0 1
			1 1 1 1
			1 1 1 0
			1 0 1 0
			1 0 1 1



### Prova adesso!

• Codice Gray

Dopo aver determinato la lunghezza del codice necessario, codificare i numeri seguenti:

1. 4 e 7
2. 15 e 22
3. 40 e 54
4. 80 e 99

## ■ 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 e 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 **Petherick** 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 configurazione.

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	1001	1001	1010	1000	1010	1101	1010
10	1111								
11	1110								
12	1010								
13	1011								
14	1001								
15	1000								

## ■ Il codice 1 su n

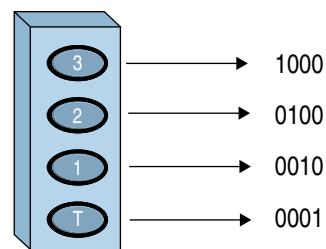
Tra i codici posizionali abbiamo visto il codice 2/5, la cui caratteristica fondamentale era quella di avere il numero di 1 costante all'interno di ogni configurazione.

Questa particolarità è presente anche nel codice **1 su n**, che associa a **ognuna delle n** possibili configurazioni **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 128 configurazioni: generalmente viene esteso e utilizzato per rappresentare gli ulteriori 6 simboli del codice esadecimale: A, B, C, D, E, F.



## Prova adesso!

- Codice a sette segmenti

Codifica in binario utilizzando un codice a sette segmenti i seguenti numeri:

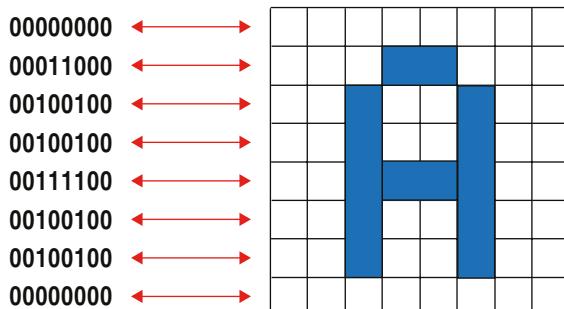
- 234
- 3590
- 2376
- ALESSIA

## ■ 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  $M \times N$  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  $9 \times 5$ .

a	b	c	d	e	f	g	h	i
A	B	C	D	E	F	G	H	I
j	k	l	m	n	o	p	q	r
J	K	L	M	N	O	P	Q	R
s	t	u	v	w	x	y	z	
S	T	U	V	W	X	Y	Z	

## ■ 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.



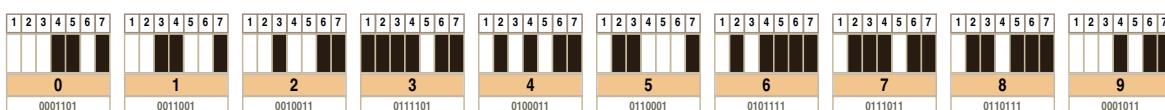
### ORIGINE DEL CODICE A BARRE

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 Settanta, 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*).

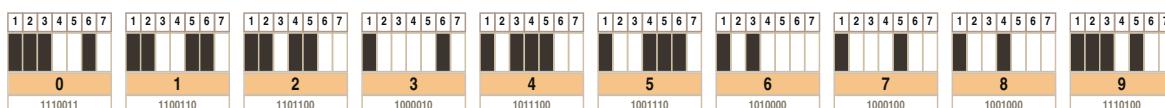
La codifica del **codice alfanumerico** avviene mediante un'alternanza di barre verticali e di spazi disposti in modo tale da essere letti semplicemente da un dispositivo ottico.

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 **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.

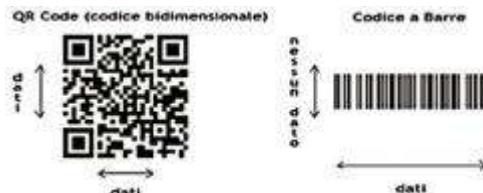


- **Prefisso EAN:** sono le prime tre cifre del codice **ISBN**, introdotte a partire dal 2007; indicano che si è in presenza di un libro (codice 978 del sistema **GS1**).
- **Gruppo linguistico:** è l'identificativo del paese o dell'area linguistica dell'editore; può utilizzare da 1 a 5 cifre (per l'Italia è 88).
- **Editore:** è l'identificativo della casa editrice o del marchio editoriale; può utilizzare da 2 a 7 cifre (per **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 **deqrificazione**.

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.

## AREA digitale



Come generare e leggere i QR Code

## ■ HCCB o Microsoft Tag

La tecnologia **HCCB** o **High Capacity Color Barcode**, è la proposta di **Microsoft Research** come una variante del **QR Code** dove, al posto dei quadratini monocromatici, vengono utilizzati gruppi di triangoli inseriti in una griglia  $5 \times 10$  o più grande: ciascun triangolo può avere quattro possibili colori.

Questo sistema permette alla tecnologia **HCCB** di codificare, nello stesso spazio, un numero maggiore di informazioni.

### ESEMPIO

Ad esempio con un quadratino bianco e nero si può memorizzare solo un bit, mentre con 4 colori nello stesso quadratino si codificano 2 bit.

Colori	Bit
Yellow	00
Purple	01
Cyan	10
Black	11

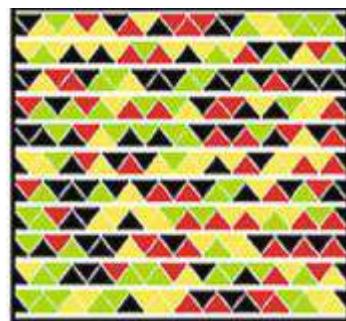
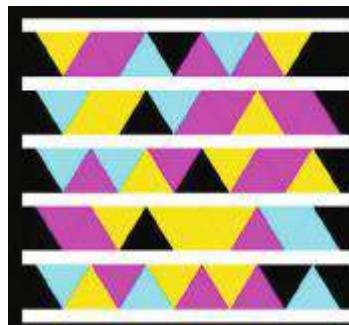
L'immagine seguente rappresenta un byte nel formato B/W e utilizzando i **HCCB**.



Utilizzando solo 8 colori si possono generare 2000 bytes binari, 3500 caratteri alfabetici in un pollice quadrato, che equivale a due pagine di testo.

La tecnologia **HCCB** posiziona intorno al tag un confine nero, circondato da un bordo bianco spesso: il contorno nero individua i bordi dell'etichetta e il bordo inferiore si riconosce per la sua dimensione doppia, così da poter individuare l'orientamento per la sua corretta decodifica.

Le immagini seguenti riportano due esempi di etichette **HCCB** di dimensioni diverse.



La tabella seguente riporta le differenze tra i QR Code e i Microsoft Tag.

	<b>QR-Code</b>	<b>Microsoft Tag</b>
Tecnologia	Quadrati bianchi e neri	HCCB (triangoli multicolore)
Quantità dei caratteri contenuti	4296 caratteri alfanumerici o 7089 caratteri numerici	3500 caratteri per pollice quadrato
Licenza	Free	Proprietà
Librerie	Free	Free (senza codice sorgente)
Personalizzazione	Non possibile	Possibile
Supporti	Carta e contenuti video	Carta stoffa e contenuti video
Connessione internet	Non necessaria	Necessaria
Contenuto	Statico	Dinamico
Architettura	Metodo diretto	Metodo indiretto

Il principale svantaggio della tecnologia HCCB è che la loro **licenza è proprietaria**: il brevetto appartiene alla **Microsoft** ed è necessario scaricare l'applicazione **Microsoft** per poterli decodificare.

## Verifichiamo le competenze

### 1. Esercizi

**1** Esegui le seguenti operazioni in eccesso 3.

$$\begin{array}{r} 3 + \dots + \\ 1 = \dots = \end{array}$$


---

$$\begin{array}{r} 9 + \dots + \\ 7 - \dots = \end{array}$$


---

xx .....

(risultato errato dato che non supera il 9)

xx .....

(risultato errato dato che supera il 9)

$$\begin{array}{r} \dots - \\ \dots = \end{array}$$

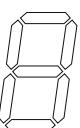

---

$$\begin{array}{r} \dots + \\ \dots = \end{array}$$

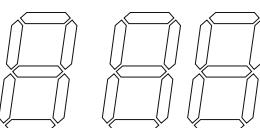

---

**2** Codifica i seguenti numeri con un codice a sette segmenti.

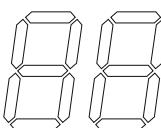
3



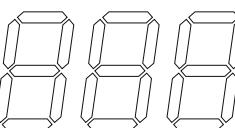
24



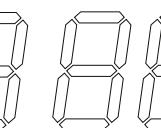
71



60

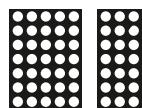


859

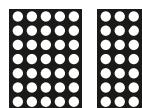


**3** Codifica i seguenti numeri/lettere con una matrice di punti 7 x 5.

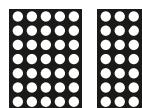
4



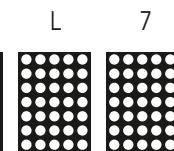
C



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.*

# La correzione degli errori

In questa lezione 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.

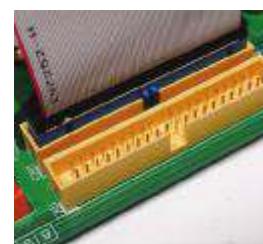


### BUS

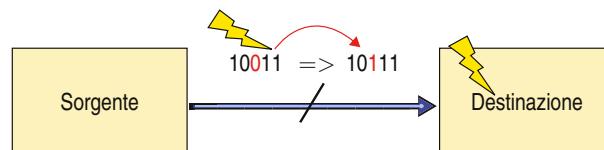
Con il termine **bus** si intende la connessione esistente tra due o più dispositivi di un PC per effettuare la trasmissione dei dati: la sua larghezza è costituita dal numero di conduttori che lo compongono e rappresenta il numero di bit che possono essere inviati contemporaneamente (**in parallelo**) 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.

AREA *digitale*



Errori e probabilità

In presenza di errore ci sono sostanzialmente quattro possibilità:

- scartare il messaggio;
  - richiedere la **ritrasmessione** 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 o distanza di Hamming



### 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:

$$D(1000,1010) = 1$$

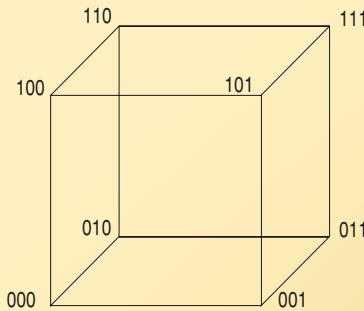
$$D(0110,0000) = 2$$

$$D(0100,1010) = 3$$

$$D(0100,0011) = 3$$

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.

### ESEMPIO

In figura è riportato il percorso per andare da 000 a 101: la strada più breve è passare da due segmenti: queste due configurazioni 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.

### ESEMPIO

Calcoliamo la distanza tra le due codifiche  $s_1$  e  $s_2$ .

$$s_1 = 10001001$$

$$s_2 = 10110001$$

$$s_1 \text{ XOR } s_2 = 00111000$$

La **distanza di Hamming** tra  $s_1$  e  $s_2$  è pari a 3.

## Distanza minima



### 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:

$$\text{DMIN (Codice ASCII)} = 1$$

in quanto esistono configurazioni che differiscono per un solo bit.

I codici non ridondanti hanno  $\text{DMIN} = 1$ . I codici ridondanti devono avere  $\text{DMIN} > 1$ .

Un codice impiegato per la rilevazione di tutti i possibili errori singoli, o **SEDC** (*Single Error Detection Code*), non deve utilizzare le configurazioni che **distano uno** da ciascuna delle configurazioni utilizzate, altrimenti un singolo errore porterebbe a una codifica valida e quindi sarebbe impossibile individuarlo.



### CODICE SEDC

Un codice SEDC deve avere almeno  $\text{DMIN} = 2$ .

## ESEMPIO

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

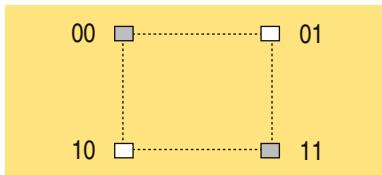
Se un bit si sporca si passa in situazioni sia accettate sia non valide: quindi non sempre è possibile riconoscere la presenza di una situazione di malfunzionamento.



### 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**.

L'aggiunta di bit di ridondanza permette di costruire codici che consentono di controllare eventuali errori di trasmissione: si hanno due tipi di codici ridondanti:

- 1 **codici a rivelazione di errore**: consentono di individuare la presenza di un errore;
- 2 **codici a correzione di errore**: consentono non solo di individuare la presenza di un errore, ma anche di identificarne la posizione in modo da poterlo correggere.

## Codice BCD con controllo di parità

Il codice più semplice che ci permette di individuare un errore è il **codice BCD con bit di parità**, che consiste nell'aggiungere un bit in più alla codifica **BCD**, detto appunto **bit di parità**, in modo tale da rendere pari il numero degli 1 presenti in ogni configurazione.

### 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	000000000
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 è 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.



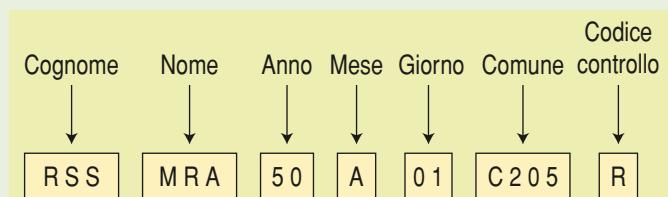
## Zoom su...

### 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.

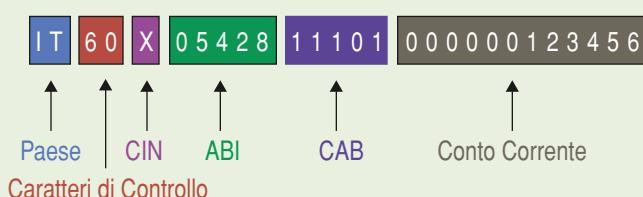


#### 2 Codice IBAN

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).

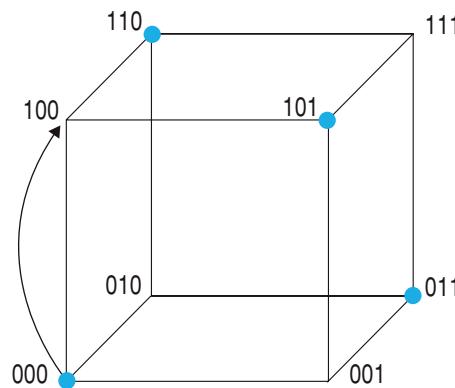
#### ESEMPIO

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.

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.

## AREA digitale

Combinazioni a distanza 2

### 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
Dato3	0	0	0	1	0	1	0	0
Dato1	1	1	0	0	1	1	1	1
Dato3	0	0	1	1	0	0	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 rilevare 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.



◀ 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:  
 $\text{bit\_dati} + \text{bit\_controllo} + 1 \leq 2^{\text{bit\_controllo}}$

#### ESEMPIO

- con 3 bit di controllo e 4 bit dei dati (1 **nibble**) avremo:  $3 + 4 + 1 \leq 2^3$ , quindi è ottimo;
- con 4 bit di controllo e 8 bit dei dati (**ASCII**) avremo:  $4 + 8 + 1 \leq 2^4$ , 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 per 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 12 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 ...

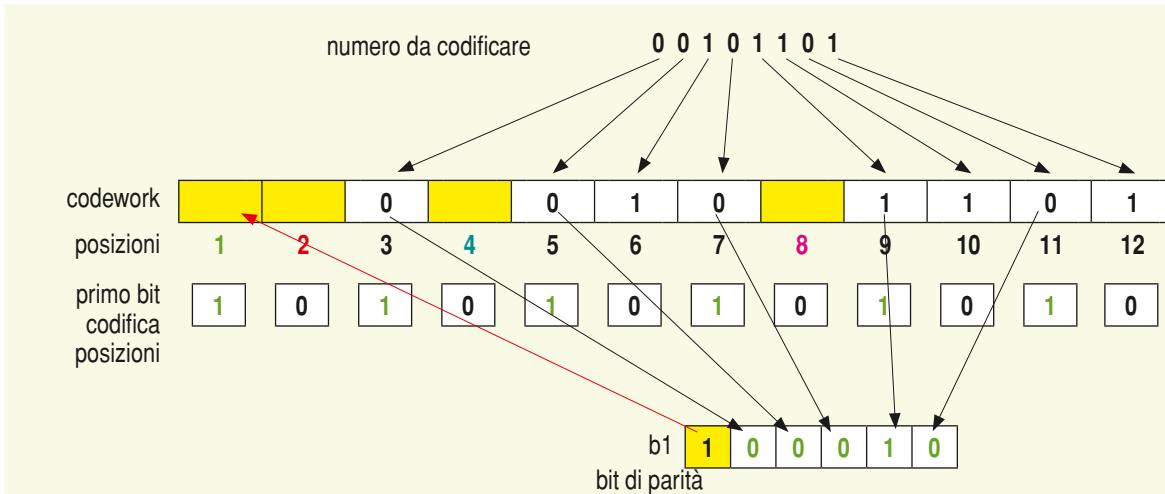
Il **quarto bit** è di parità per tutti i bit in cui la codifica binaria della posizione ha il 4° bit = 1 cioè

8 9 10 11 12 ...

### ESEMPIO

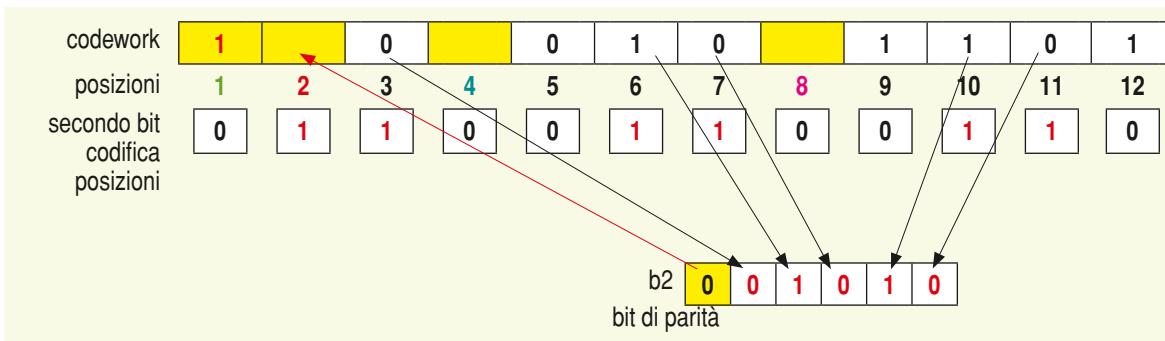
Vediamo un esempio di codifica della parola 00101101 nel **codice di Hamming** (12.8).

Iniziamo a determinare il **primo** bit di parità (posizione 2<sup>0</sup>) quello che andrà inserito nella posizione 1, ottenuto dalla parità dei bit che hanno un 1 nel **primo** bit della codifica binaria che ne individua la loro posizione cioè 3,5,7,9,11.



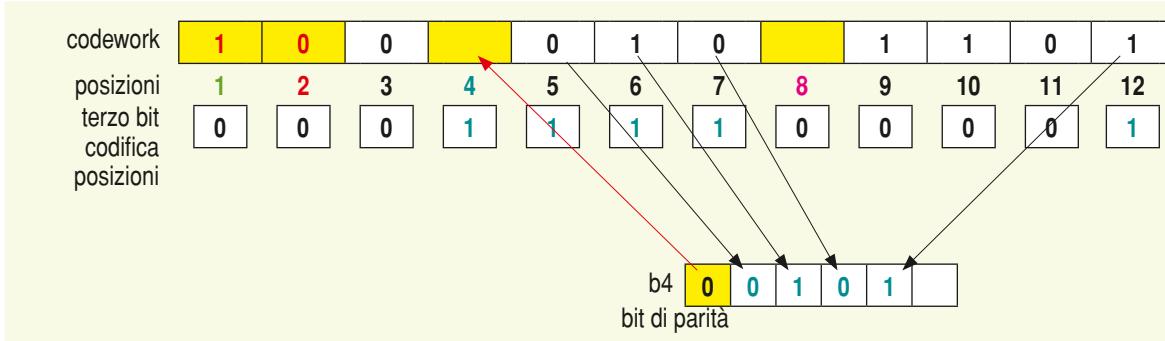
Essendoci un solo uno la parità si ottiene ponendo  $b1 = 1$ .

Analogo il procedimento per il **secondo** bit di parità, cioè il bit2 (posizione  $2^1$ ), ottenuto come parità della somma dei bit che hanno un 1 nel **secondo** bit della codifica binaria che ne individua la loro posizione cioè 3,6,7,10,11:



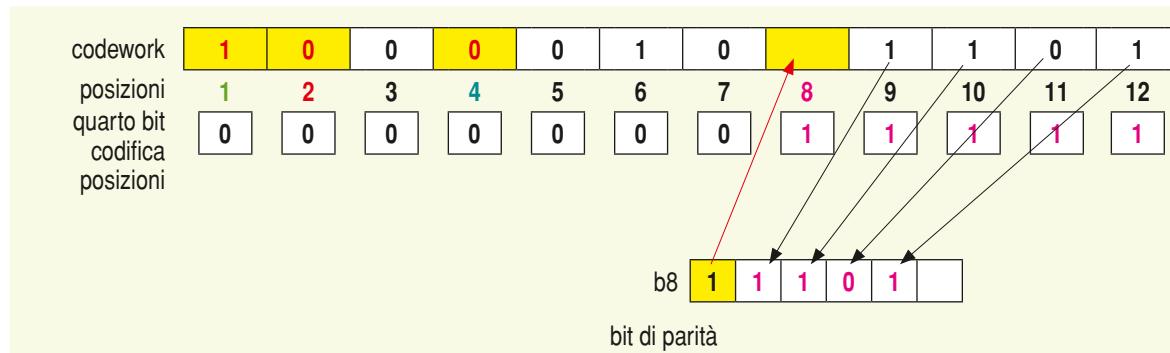
Essendoci due 1 la parità è già “fatta” e quindi  $b2 = 0$ .

Analogo il procedimento per il **terzo** bit di parità, cioè il bit4 (posizione  $2^2$ ), ottenuto come parità della somma dei bit che hanno un 1 nel **terzo** bit della codifica binaria che ne individua la loro posizione cioè 5, 6, 7, 12:



Essendoci due 1 la parità è già “fatta” e quindi anche  $b4 = 0$ .

Analogo il procedimento per il quarto bit di parità, cioè il bit<sub>8</sub> (posizione  $2^3$ ), ottenuto come parità della somma dei bit che hanno un 1 nel quarto bit della codifica binaria che ne individua la loro posizione cioè 9, 10, 11 e 12:



Essendoci tre 1 la parità si ottiene ponendo  $b_8 = 1$ .

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.

### ESEMPIO

Vediamo un esempio di individuazione e correzione di un errore: supponiamo di aver ricevuto il codice 00101101 e lo analizziamo per verificarne la correttezza dei bit:

dato	00101101												
3210	000	000	100	100	0100	010	010	010	100	100	100	100	
codifica	1	0	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

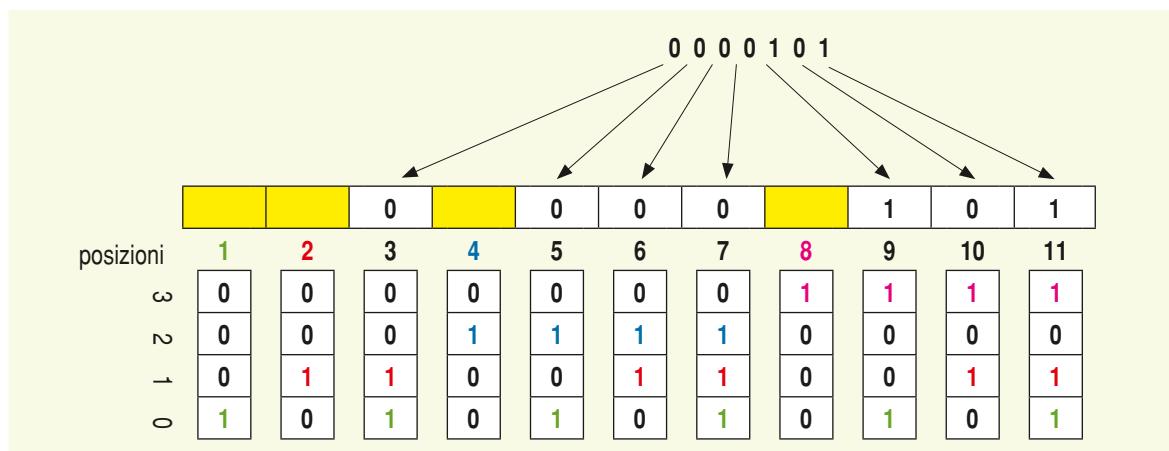
Sommendo le posizioni dei bit con check errato, cioè il secondo e il quarto, otteniamo la posizione del bit del messaggio che si è sporcato: il bit numero 6.

Come secondo esempio effettuiamo una codifica con **codice di Hamming** dispari (11/4) e poi individuiamo eventuali errori dopo la sua trasmissione analizzando i dati ricevuti.

### ESEMPIO

Codifichiamo per la trasmissione la seguente stringa di 7 bit: 0000101.

#### A 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	1
b4	1	0	0	0	0	0
b8	1	1	0	1		

Il **codework** ottenuto è quindi: **10010001101**.

#### B 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. Risposta multipla

**1** Qual è la distanza di Hamming tra le seguenti due codifiche (a 8 bit)?

00101000 e 00011001

- a. 0
- b. 1
- c. 2
- d. 3
- e. 4
- f. 5

**2** Qual è la distanza minima di Hamming tra le seguenti due codifiche (a 8 bit)?

00101000, 00011001

- a. 0
- b. 1
- c. 2
- d. 3
- e. 4
- f. 5

**3** Quale delle seguenti configurazioni è errata in parità pari?

- a. 1000100
- b. 0000111
- c. 0001100
- d. 1010101
- e. 1010101
- f. 1010110

**4** Quale delle seguenti configurazioni è errata in parità dispari?

- a. 0000100
- b. 1000111
- c. 1001100
- d. 0010101
- e. 1010011
- f. 1010100

**5** Se hai ricevuto il byte 00001111 e un bit si è sporcato, qual è il messaggio inviato?

- a. 0000100
- b. 0000111
- c. 0001100
- d. 0010101
- e. 0110101
- f. 11001111

**6** Se hai ricevuto il byte 00001111 e due bit si sono sporcati, qual è il messaggio inviato?

- a. 0010100
- b. 0000111
- c. 0001100
- d. 0010101
- e. 0011101
- f. 00001110

**7** Qual è la codifica di Hamming (7,4) del messaggio

1 0 0 1?

- a. 0010101
- b. 0010011
- c. 0011000
- d. 0011001
- e. 0010111
- f. 0011111

**8** Quale è la codifica di Hamming (12/4) del messaggio 1010 1010?

- a. 1011 0100 1010
- b. 1110 0100 1010
- c. 1111 0101 1010
- d. 1111 0100 1010
- e. 1111 0100 1110
- f. 1111 0100 1111

**9** Quale messaggio è codificato nella parola in codice di Hamming (12,4) 1011 1000 1101 dove un bit si è sporcato?

- a. 0100 1100
- b. 1100 1100
- c. 1100 1000
- d. 1110 1100
- e. 1100 0100
- f. 1100 0110

**10** Quale messaggio è codificato nella parola in codice di Hamming (17,12) 1001 1010 1001 1010 1 dove un bit si è sporcato?

- a. 0101 1001 1011
- b. 0101 0001 1010
- c. 0101 1011 1010
- d. 0101 1001 1110
- e. 0101 1001 1010
- f. 0101 1001 1111

## Verifichiamo le competenze

### 1. Esercizi

- 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:

$$d_{12} = \underline{\hspace{2cm}} \quad d_{23} = \underline{\hspace{2cm}} \quad d_{13} = \underline{\hspace{2cm}}$$

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:

$$d_{12} = \underline{\hspace{2cm}}$$

$$d_{23} = \underline{\hspace{2cm}}$$

$$d_{13} = \underline{\hspace{2cm}}$$

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 .....<sub>2</sub>

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:

r1	r2	m1	r3	m2	m3	m4
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:

r1	r2	m1	r3	m2	m3	m4
.....	.....	.....	.....	.....	.....	.....

La parola codificata è: m<sub>1</sub>m<sub>2</sub>m<sub>3</sub>m<sub>4</sub> = .....

# ESERCITAZIONI DI LABORATORIO 1

## SIMULARE UNA TRASMISSIONE SERIALE CON EXCEL

### ■ La comunicazione

Fondamentalmente il computer comunica con le periferiche attraverso due tipi di trasmissione che possono essere così sintetizzate:

- comunicazione parallela;
- comunicazione seriale.

Il primo tipo di trasmissione è assai dispendioso di risorse in quanto utilizza un cavo per ciascun bit da trasmettere. Negli ultimi anni la trasmissione seriale si è affermata come uno standard di comunicazione adottato ad esempio in alcuni standard come le porte USB (*Universal Serial Bus*).

La comunicazione seriale avviene tramite un unico mezzo trasmissivo in cui i dati vengono trasmessi e ricevuti serialmente, ossia un bit alla volta. Consideriamo a titolo di esempio la comunicazione seriale del carattere esadecimale 25h. In binario questo carattere assume la seguente forma:

**00100101**

Questa sequenza di bit verrà trasmessa su cavo seriale, un bit alla volta, come di seguito:

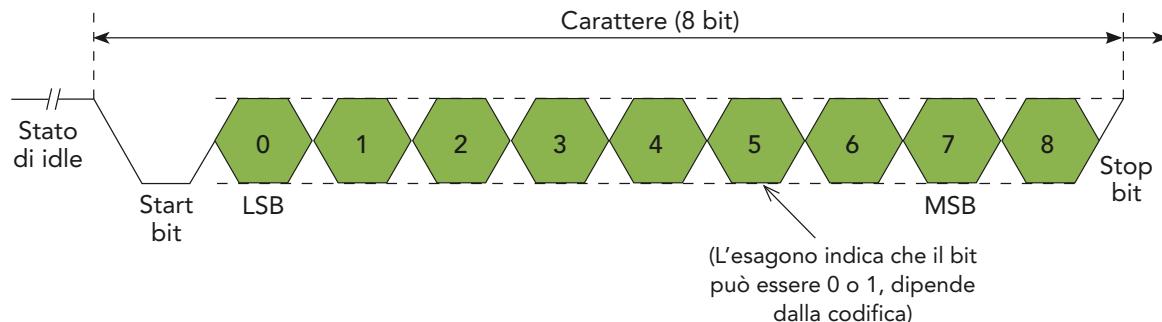


Esistono due tipi di **comunicazione seriale**:

- **sincrona**;
- **asincrona**.

La **trasmissione sincrona** associa l'invio o la ricezione di ogni bit a un impulso di **clock**, mentre la **trasmissione asincrona** non dispone di un impulso di **clock** associato ai bit trasmessi. Quest'ultima è di gran lunga la più diffusa. Nel trasferimento seriale asincrono, i bit

da trasferire si dividono in blocchi o pacchetti, che vengono spediti preceduti da un segnale di inizio (**START**), seguito da un segnale di fine (**STOP**). La figura seguente mostra la trasmissione asincrona di 8 bit di dati, preceduti dal bit di **START** e terminanti con il bit di **STOP**:



Lo standard utilizzato nella comunicazione seriale dei computer si chiama **RS-232**. Le caratteristiche principali di questo standard sono le seguenti:

- 1 possiamo specificare se inviare 8 bit oppure 7 per ogni **carattere**;
- 2 possiamo utilizzare un particolare bit denominato **bit di parità**, che viene utilizzato per verificare la correttezza dei dati ricevuti. Il **bit di parità pari** vale 1 se il numero di bit che valgono 1, nel carattere inviato, è dispari. Il ricevente a sua volta conterà i bit pari a uno nel carattere, e in caso di discrepanza segnalera un errore;
- 3 il bit di stop serve per evidenziare il termine della sequenza di bit. Non a caso esso viene aggiunto in coda alla sequenza di trasmissione;
- 4 la velocità di trasmissione, chiamata **baud rate** può andare dai 300 bit/s fino a 115200 bit/s.

### ESEMPIO *Simulare una trasmissione seriale con Excel*

Vediamo come realizzare in Excel un simulatore di trasmissione seriale. Nel foglio in questione ([UA2\\_Lab\\_1 La trasmissione seriale.xlsx](#)) l'utente inserisce i codici binari da trasmettere nelle celle C7:L7:



Nella cella N7 abbiamo la formula che calcola il codice **ASCII** del carattere da trasmettere. Prima di tutto la cella contiene una funzione logica =SE che verifica se i caratteri sono stati

immessi, in caso contrario colloca nella cella **N7** il valore Null (""). Per fare questo utilizziamo l'operatore logico di disgiunzione (**Or**), rappresentato dalla funzione **O()**, all'interno della condizione della funzione **=SE**, nel modo seguente:

```
=SE(O(D7="" ;E7="" ;F7="" ;G7="" ;H7="" ;I7="" ;J7="" ;K7="" ) ;"" ;...)
```

È sufficiente che una sola cella tra **D7** e **K7** sia uguale a null (= "") per restituire Null nella cella risultato. Come vediamo la formula è incompleta, la parte che tratteremo ora è stata sostituita dai puntini.

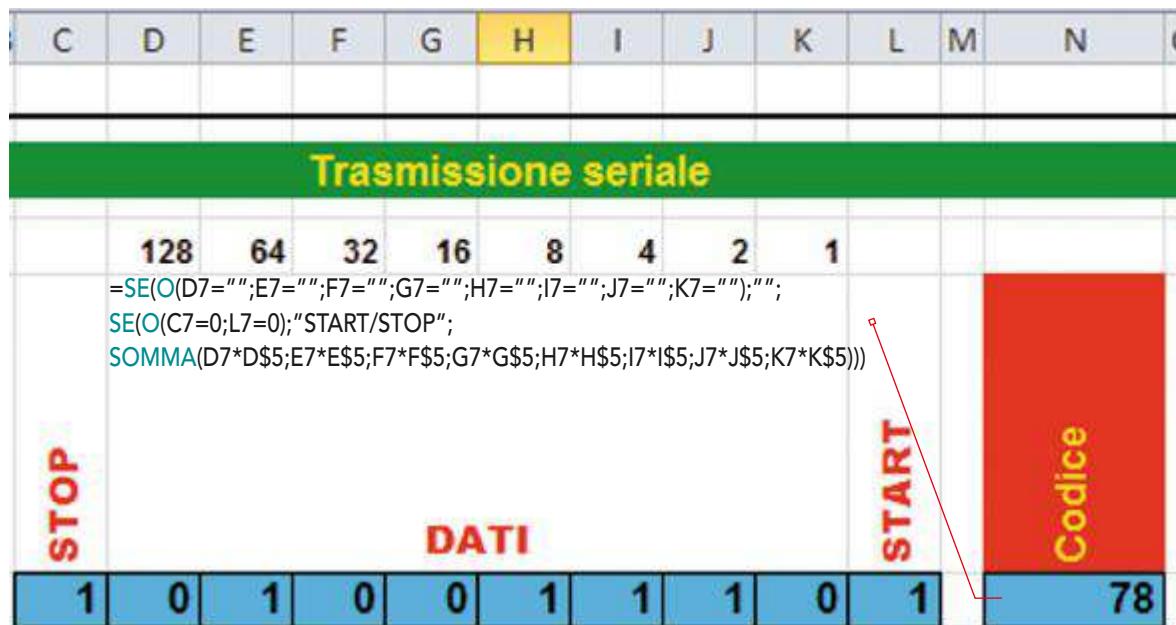
Il resto della funzione **=SE** agisce solo quando tutti i bit sono stati riempiti con un valore logico (0 o 1). Prima di tutto viene effettuata la verifica sulle celle di **START** e **STOP** (**C7** e **L7**) mediante un'altra funzione condizionale (**=SE**):

```
... ; SE(O(C7=0;L7=0); "START/STOP"; ...)
```

Quando anche una sola delle celle (**C7** e **L7**) contiene zero, apparirà nella cella **N7** la dicitura **"START/STOP"** per indicare che mancano i bit di marcatura. Vediamo adesso la parte finale della formula rappresentata qui sopra dai puntini di sospensione. In questa sezione, quando cioè i codici di START e STOP sono corretti, inseriamo il polinomio di calcolo del valore in decimale degli 8 bit presenti nelle celle da D7 a K7:

```
... ; SOMMA(D7*D$5;E7*E$5;F7*F$5;G7*G$5;H7*H$5;I7*I$5;J7*J$5;K7*K$5)) )
```

Vediamo la formula completa da collocare nella cella **N7**:



Come possiamo notare appare il codice ASCII del numero binario rappresentato nelle celle da D7 a K7.

Dopo aver copiato la cella **N7** nelle celle successive, in modo da poter simulare l'invio di più caratteri, passiamo all'inserimento della formula nella cella **A7**. In tale cella vogliamo far apparire il carattere da inviare. Per fare questo usiamo una funzione di Excel assai comoda e di facile utilizzo, si tratta di **=CODICE.CARATT()** che viene utilizzata all'interno di una funzione condizionale che determina se la cella da convertire (N7), contiene un codice valido. Per fare questo utilizza la formula seguente:

```
=SE(0(N7="" ; N7="START/STOP" ; N7=0) ; "" ; CODICE.CARATT(N7))
```

Come possiamo notare il codice presente nella cella **N7** viene convertito nel relativo carattere **ASCII** solo quando la cella **N7** è diversa da zero, da "START/STOP" e da Null (**""**).

Il risultato finale è il seguente, inserendo le sequenze di caratteri indicate viene trasmessa la parola "informatica":

Trasmissione seriale												
Carattere / Significato		peso bit										
		128	64	32	16	8	4	2	1	START	STOP	Codice
i		1	0	1	1	0	1	0	0	1	1	105
n		1	0	1	1	0	1	1	1	0	1	110
f		1	0	1	1	0	0	1	1	0	1	102
o		1	0	1	1	0	1	1	1	1	1	111
r		1	0	1	1	1	0	0	1	0	1	114
m		1	0	1	1	0	1	1	0	1	1	109
a		1	0	1	1	0	0	0	1	1	1	97
t		1	0	1	1	1	0	1	0	0	1	116
l		1	0	1	1	0	1	0	0	1	1	105
c		1	0	1	1	0	0	0	1	1	1	99
a		1	0	1	1	0	0	0	1	1	1	97

Parola inviata

informatica



Prova adesso!

- Utilizzare Excel per simulare la trasmissione seriale asincrona



APRI IL FILE UA2\_Lab\_1 La trasmissione seriale

Nell'esempio proposto l'utente inserisce manualmente i valori binari dei caratteri da trasmettere. L'utente deve invece immettere il carattere **ASCII** nella colonna **A** e il foglio di calcolo deve calcolare il valore binario di ciascun carattere nelle celle di colonne **D..K**.

## ESERCITAZIONI DI LABORATORIO 2

# CALCOLO DEL CHECK DIGIT NEI CODICI EAN 8 ED EAN 13

### ■ Caratteristiche del codice EAN

Il codice EAN (*European Article Number*) è un formato di codice a barre usato prevalentemente in Europa per la marcatura dei prodotti destinati alla vendita al dettaglio, così da permettere la lettura ottica e facilitare tutte le operazioni di gestione informatizzata degli articoli permettendo l'identificazione univoca di prodotti destinati al consumatore finale.

Possiamo riassumere le principali caratteristiche in:

- è un codice pluridimensionale: sono presenti 4 diversi spessori di barre/spazi;
- ogni carattere del codice è codificato come usuale in modo binario;
- ogni carattere è codificato usando 7 moduli;
- ogni modulo ha uno spessore fisso di 0.33 mm;
- ognuno multiplo del modulo unitario;
- è continuo, cioè sono significativi sia gli spazi che le barre;
- è leggibile nei 2 sensi, e può rappresentare un numero fisso di caratteri;
- il set di caratteri normalmente usato per la parte numerica è il ◀ OCR-B ▶.



◀ **OCR (Optical Characters Recognition)** è la modalità più diffusa di lettura ottica di documenti e moduli: il font di caratteri utilizzato è stato appositamente progettato affinché un apposito programma **OCR** ne effettui il riconoscimento in modo da trasformare documenti stampati in file di caratteri. ►

Esistono 2 versioni principali di codice EAN:

1. l'**EAN 13** (o “normale”);

2. l'**EAN 8** (o “ridotto”).



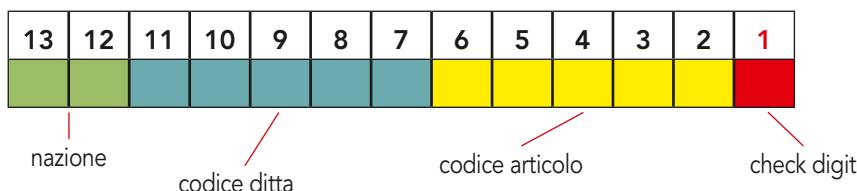
## ■ Interpretazione del codice EAN

Il codice **EAN** è un codice composto, dove sono presenti 4 gruppi di cifre:

- le prime due cifre del numero indicano la **nazione di produzione**;

Cifre	Nazione
00-09	USA, Canada
30-37	Francia
40-43	Germania
45	Giappone
50	Gran Bretagna
76	Svizzera
80-83	Italia
87	Olanda
90-91	Austria

- il secondo gruppo composto dalle cifre seguenti indica la ditta di produzione, assegnato dall'**istituto nazionale** preposto alla codifica;
- il terzo gruppo composto anch'esso da 5 cifre successive indicano univocamente il tipo di prodotto, assegnato dalla **ditta produttrice**;
- l'ultima cifra è quella di **controllo**.



La presenza di un codice di controllo rende il codice **EAN** un codice rilevatore di errore.

Un codice **rivelatore di errori** è tale che cambiando un solo simbolo in una qualsiasi parola, la n-pla che si ottiene non è una parola del codice.

In questo modo la generazione di un errore su una parola appartenente al codice produce una parola non appartenente al codice e viene quindi individuata come errata.

### ESEMPIO

Nell'esempio seguente è possibile individuare i quattro elementi che compongono il codice EAN13:

Codice nazione: Italia  
 Codice Ditta: 32089  
 Codice prodotto: 00001  
 Cifra di controllo: 7



## ■ Calcolo del check digit

Descriviamo ora l'algoritmo che permette di ricavare la **cifra di controllo** o **check digit**, che sia nella versione 13 che in quella 8 è il carattere più a destra.

Per le determinazione di tale cifra è comodo numerare i singoli carattere da destra verso sinistra, in modo che il **check digit** sia in posizione 1, come indicato nella figura precedente.

I passi della procedura di calcolo sono i seguenti:

1. si **sommano** i valori dei caratteri in **posizione pari**;
2. il risultato così ottenuto si **moltiplica per 3**;
3. si **sommano** i valori dei caratteri in **posizione dispari** a partire dalla posizione 3;
4. si sommano i due risultati parziali;
5. come **check digit** si assume il numero da sommare al risultato precedente per ottenere un multiplo di 10.

Se osserviamo le prime due operazioni equivale considerare con peso 3 le cifre disposizione pari: quindi possiamo riscrivere l'algoritmo in questo modo:

1. si associa un peso a ogni cifra:
  - peso 3 ai valori dei caratteri in posizione pari partendo dalla posizione 2
  - peso 1 ai valori dei caratteri in posizione dispari partendo dalla posizione 3
2. si moltiplica ogni cifra per il suo peso
3. si sommano i prodotti
4. si calcola il **check digit** individuando il numero da sommare al risultato precedente per ottenere un multiplo di 10

### ESEMPIO

Calcoliamo la cifra di controllo del seguente codice EAN 13



Individuiamo la posizione di ogni cifra:

13	12	11	10	9	8	7	6	5	4	3	2	1
4	2	7	6	2	2	1	3	5	7	4	6	?

1. sommiamo le cifre di posto pari:  $6 + 7 + 3 + 2 + 6 + 2 = 26$
2. moltiplichiamo per 3:  $26 \cdot 3 = 78$
3. sommiamo le cifre di posto dispari:  $4 + 5 + 1 + 2 + 7 + 4 = 23$
4. sommiamo i due parziali:  $78 + 23 = 101$
5. la "decina più vicina" è 110, quindi:  $110 - 101 = 9$  (**cifra di controllo**)

Come secondo esempio verifichiamo la correttezza della cifra di controllo del seguente codice:



Individuiamo la posizione di ogni cifra:

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

1. sommiamo le cifre di posto pari:  $6 + 4 + 0 + 8 + 6 + 5 = 29$
2. moltiplichiamo per 3:  $29 * 3 = 87$
3. sommiamo le cifre di posto dispari:  $4 + 2 + 9 + 7 + 5 + 4 = 31$
4. sommiamo i due parziali:  $87 + 31 = 118$
5. la “decina più vicina” è 120, quindi:  $120 - 118 = 2$  [OK]

## ■ Verifica del check digit in Excel

Il calcolo del **check digit** può semplicemente essere fatto con un foglio elettronico. Realizziamo una semplice interfaccia come quella riportata nella figura seguente:

A	B	C	D	E
1				
2				
3				
4			Check EAN 13	
5				
6		Codice		
7				
8				
9		chekdigit		
10				
11		controllo		
12				
13				
14				

Nella prima cella D6 verrà digitato il codice **EAN 13** da verificare.

Nella seconda cella D9 generiamo il **check digit** mediante la seguente formula:

The formula in the formula bar is: `=RESTO(10-RESTO(SOMMA(--STRINGA.ESTRAI(D6;{2\4\6\8\10\12};1)*3; STRINGA.ESTRAI(D6;{1\3\5\7\9\11};1)*1);10);10)`

che è composta da quattro parti, come i passi dell'algoritmo:

- STRINGA.ESTRAI(D6;{2\4\6\8\10\12};1)\*3 estrae cifre pari e le moltiplica per 3
- STRINGA.ESTRAI(D6;{1\3\5\7\9\11};1)\*1 estrae cifre dispari e le moltiplica per 1
- somma i risultati parziali in <somma>
- RESTO(10-RESTO(<somma>;10);10) calcola due volte il resto del numero diviso 10

Verifichiamo l'ultima formula:

- prendiamo il valore della somma (101) dell'esercizio precedente
- calcoliamo RESTO(101;10), cioè il resto di 101 diviso 10 che è 1
- calcoliamo RESTO( $10 - 1; 10$ ) = RESTO (9;10) = 9 che è il **check digit** cercato.

Nella terza cella [D13] confrontiamo il **check digit** ottenuto con l'ultimo carattere del codice presente in D6 con la seguente formula:



I seguenti due esempi mostrano il risultato nel caso di un codice corretto e di uno errato.

Check EAN 13	
Codice	1234567890128
chekdigit	8
controllo	VERO

Check EAN 13	
Codice	2234567890128
chekdigit	?
controllo	FALSO



### Prova adesso!

Dopo aver verificato la correttezza o meno dei seguenti codici **EAN13**:



Realizza un foglio elettronico che effettua il controllo per la versione **EAN 8** come indicato nella seguente figura.

Check EAN 8	
Codice	?????????
chekdigit	???
controllo	???

Quindi verifica la tua soluzione con quella disponibile nella sezione materiali della sezione riservata a questo volume all'indirizzo [www.hoepliscuola.it](http://www.hoepliscuola.it) nel file **UA2\_lab2\_check\_EAN**.

# ESERCITAZIONI DI LABORATORIO 3

## UN PROGRAMMA PER LA CODIFICA DI HAMMING

### ■ Codici di Hamming per la identificazione e la correzione di errore

Sappiamo che se la distanza di Hamming tra codici legittimi è pari almeno a  $d = t + 1$ , allora è identificabile l'errore sino all'ordine  $t$  di errori (dove  $t$  è un intero).

Se invece la distanza di Hamming tra codici legittimi è pari almeno a  $d = 2t + 1$ , allora è correggibile l'errore sino all'ordine  $t$  di errori (dove  $t$  è un intero).

In questa lezione realizzeremo con un foglio elettronico il caso di codice di Hamming autocorrettivo per bit singolo, in grado cioè di correggere un eventuale errore su un solo bit; in particolare il codice 12/4, cioè dove il messaggio da trasmettere è composto da 8 bit ai quali saranno aggiunti 4 bit di parità e grazie a questi bit di parità sarà possibile individuare la presenza di un errore di trasmissione e di effettuarne la correzione.

Per prima cosa predisponiamo la parte superiore del foglio di calcolo dove andremo a inserire la parola da trasmettere e avremo la generazione dei bit di parità:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2																
<b>Generazione codice di Hamming pari</b>																
3																
4																
5	parola															
6																
7				c1	c2	b3	c4	b5	b6	b7	c8	b9	b10	b11	b12	
8	trasmesso															

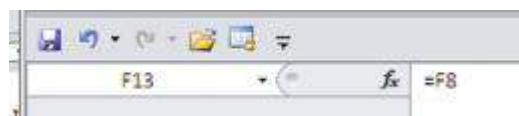
Realizziamo una codifica a parità pari, dove cioè i bit che aggiungiamo rendono pari i bit che controllano: i bit da generare sono **c1** in posizione **E8**, **c2** in posizione **E8**, **c4** in posizione **G8** e **c8** in posizione **K8**.

Predisponiamo la parte necessaria per la determinazione dei singoli bit di parità aggiungendo 4 righe, dove andremo a riprendere i bit che volta per volta concorreranno alla determinazione della parità.

10	c8->												
11	c4->												
12	c2->												
13	c1->												
14		1	2	3	4	5	6	7	8	9	10	11	12

Ogni riga genererà un bit di parità: individuiamo per ciascuna le celle coinvolte evidenziandole con lo **sfondo azzurro** mentre andremo a collocare il bit di parità nelle celle con lo **sfondo arancione**.

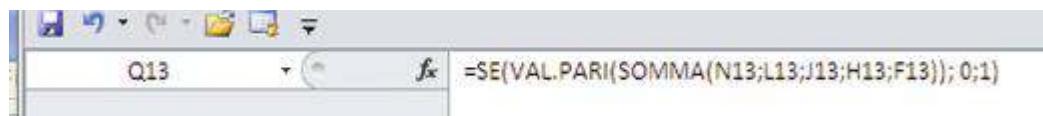
Come unica operazione andremo a “ricopiare” i bit della nostra parola da codificare in tutte le celle con lo sfondo azzurro, semplicemente con una operazione del tipo di quella fatta per la cella **F13** di sotto riportata come esempio:



Completiamo il nostro foglio con la sezione che esegue il calcolo del bit di parità per ciascuna riga:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
7																
8	trasmesso		c1	c2	b3	c4	b5	b6	b7	c8	b9	b10	b11	b12		
9																
10		c8->														
11		c4->														
12		c2->														
13		c1->														
14			1	2	3	4	5	6	7	8	9	10	11	12		

La formula che viene ripetuta per ogni riga è semplicemente data dalla somma delle celle con sfondo azzurro, come ad esempio quella riportata di seguito che calcola il c1 valutando la parità o meno della somma dei bit di posizione N13+L13+J13+H13+F13 in modo da assegnare alla cella **Q13** 0 oppure 1.



Dopo aver scritto la funzione per tutte e quattro le celle della sezione Parità (**Q10**, **Q11** e **Q12**), completiamo il nostro foglio riportando i bit di parità anche nella riga 8 dove stiamo costruendo la parola in codice.

Inseriamo una parola di esempio, come 01001101 e otteniamo quanto riportato in figura:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2																	
3																	
4																	
5	parola		0	1	0	0	1	1	1	0	1						
6																	
7			c1	c2	b3	c4	b5	b6	b7	c8	b9	b10	b11	b12			
8	trasmesso		0	1	0	0	1	0	0	1	1	1	0	1			
9																	
10	c8-->									1	1	1	0	1			
11	c4-->					0	1	0	0					1			
12	c2-->		1	0				0	0			1	0				
13	c1-->		0	0		1		0		1		0					
14			1	2	3	4	5	6	7	8	9	10	11	12			



## Prova adesso!

- Codifica di Hamming
- Rilevazione e correzione di un errore

Completa il foglio elettronico aggiungendo una sezione che riporta il codice ricevuto in trasmissione e ne effettua la verifica individuando la posizione di un eventuale errore come riportato in figura:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2																			
3																			
4																			
5	parola		0	1	0	0	1	1	1	0	1								
6																			
7			c1	c2	b3	c4	b5	b6	b7	c8	b9	b10	b11	b12					
8	trasmesso		0	1	0	0	1	0	0	1	1	1	0	1					
9																			
10	c8-->									1	1	1	0	1					
11	c4-->					0	1	0	0					1					
12	c2-->		1	0				0	0			1	0						
13	c1-->		0	0		1		0		1		0							
14			1	2	3	4	5	6	7	8	9	10	11	12					
15																			
16	ricevuto		1	1	0	0	1	0	0	1	1	1	0	1					
17																			
18																			

Quindi verifica la tua soluzione con quella disponibile nella sezione materiali della sezione riservata a questo volume all'indirizzo [www.hoepiscuola.it](http://www.hoepiscuola.it) nel file **UA2\_lab3\_HammingCode.xlsx**.

Infine realizza un foglio elettronico per la codifica 16/5.

# 3

# La codifica dei numeri

- L1 Operazioni tra numeri binari senza segno
- L2 Numeri binari relativi
- L3 Numeri reali in virgola mobile

## Esercitazioni di laboratorio

- ① Facciamo il complemento a uno e a due con Excel;
- ② L'addizione e la sottrazione in base binaria

### Conoscenze

- Sistema di numerazione decimale, binario, ottale, esadecimale
- Acquisire la nozione di complemento di un numero
- Acquisire il concetto di overflow
- 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

### Competenze

- Codificare immagini, suoni e filmati
- Codificare e decodificare numeri e codici
- Codificare i numeri in modulo e segno
- Codificare e decodificare i numeri in IEEE-P754
- Codificare un numero periodico

### Abilità

- Eseguire il complemento a 1 e a 2 di un numero binario
- Effettuare le operazioni algebriche tra numeri binari
- Convertire numeri e codici rappresentati secondo sistemi diversi
- Rappresentare i numeri in complemento a 1, a 2 e a n
- Rappresentare i numeri decimali in virgola mobile
- Utilizzare il foglio elettronico per effettuare le operazioni binarie

## AREA *digitale*



- ▶ Circuito sommatore
- ▶ Float nel turbo C++ e Dev-Cpp
- ▶ Intervalli di rappresentazione
- ▶ Esercizi per il recupero
- ▶ Esercizi per l'approfondimento



Soluzioni (prova adesso, esercizi, verifiche)  
Puoi scaricare il file anche da [hoepliscuola.it](http://hoepliscuola.it)

# Operazioni tra numeri binari senza segno

In questa lezione 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:

- CA1(A) oppure CP1(A) o semplicemente C1(A) = completamento a 1 di A;
- $\bar{A}$  = complemento a 1 di A;
- $\bar{\bar{A}}$  = 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:

$$\text{CA1}(01110) = 10001$$

$$\text{CA1}(10001) = 01110$$

$$\text{CA1}(01101101) = 10010010$$

Il numero rappresentato in complemento a 1 può essere indicato per completezza con il pedice:

$$\text{CA1}(01101100) = 10010011_{\text{CA1}}$$

oppure semplicemente

$$\text{CA1}(11101101) = 00010010_{\text{C1}}$$

## ■ 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 un esempio per ciascuna tra le tre notazioni prima indicate:

$$\text{CA2}(01110) = \text{CA1}(01110) + 00001 = 10001 + 00001 = 10010_{\text{CA2}}$$

$$\text{CP2}(01100) = \text{CP1}(01100) + 00001 = 10011 + 00001 = 10100_{\text{CP2}}$$

$$\text{C2}(10001) = \text{C1}(10001) + 00001 = 01110 + 00001 = 01111_{\text{C2}}$$

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:

$\text{CA2}(100101) \Rightarrow 011011_{\text{CA2}}$ : 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;

$\text{CA2}(10010100) \Rightarrow 01101100_{\text{CA2}}$ : 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 anch'esso.

Procediamo invertendo i bit successivi alla loro sinistra.



### Prova adesso!

- Complemento a 1
- Complemento a 2

Esegui il complemento a 2 dei seguenti numeri prima passando attraverso il complemento a 1 e successivamente verificando il risultato applicando la regola pratica appena descritta:

$$\text{CA2}(1001\ 0010) = \dots \text{CA1} + 0000\ 0001 = \dots \text{CA2}$$

$$\text{CA2}(1000\ 1110) = \dots \text{CA1} + 0000\ 0001 = \dots \text{CA2}$$

$$\text{CA2}(1111\ 0111) = \dots \text{CA1} + 0000\ 0001 = \dots \text{CA2}$$

## ■ Addizione

Riportiamo le quattro possibili combinazioni che otteniamo dalla somma tra due bit:

<b>+</b>	<b>b<sub>1</sub></b>	<b>b<sub>2</sub></b>	<b>=</b>	<b>addizione</b>
	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:

<b>binario</b>	
riporto	
	1 0 0 +
	1 1 =
	—————
	1 1 1

<b>decimale</b>	
riporto	
	4 +
	3 =
	—————
	7

<b>binario</b>	
riporto	1 1
1	1 1 0 +
	1 1 =
	—————
	1 0 0 1

<b>decimale</b>	
riporto	
	6 +
	3 =
	—————
	9

<b>binario</b>	
riporto	1 1 1
1	1 1 1 0 +
	1 1 =
	—————
	1 0 0 0 1

<b>decimale</b>	
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		decimale	
riporto	1 1 1 1	riporto	
	1 1 1 0 1 0 0 0 +		2 3 2 +
	0 1 1 0 1 1 1 0 =		1 1 0 =
	—————		—————
	0 1 0 1 0 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  $2^8 = 256$ , 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.

## AREA digitale



Circuito sommatore

## Sottrazione

Riportiamo nella tabella della verità rappresentata a lato le quattro possibili combinazioni che otteniamo dalla sottrazione tra due singoli bit:

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: ►

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: avrà bisogno di prelevare un prestito dal bit di posizione due  $b_2$ .

$b_1 - b_2$	sottrazione
0 - 0 = 0	
0 - 1 = 1	con prestito = 1
1 - 0 = 1	
1 - 1 = 0	

binario
1 1 1 -
1 1 =
—————
1 0 0

$b_2 \ b_1 \ b_0$
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.

Il risultato della sottrazione del 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	=
		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  $11 - 1 - 1 = 10 - 1 = 1$

$b_3$	$b_2$	$b_1$	$b_0$	
1	1	1	0	-
prestito	1	1		-
0	0	1	1	=
		0	1	1

$b_3$	$b_2$	$b_1$	$b_0$	
	1	1	0	-
prestito		1		-
	0	0	1	=
		1	0	1

Bit  $b_2$   $1 - 1 - 0 = 0$

Bit  $b_3$   $1 - 0 = 1$

Il risultato della sottrazione è il numero binario  $1011 = 11_{10}$ .

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  $2^8 = 256$ , 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**).



## Prova adesso!

- Addizione binaria
- Sottrazione binaria

Effettuare la conversione in binario e le operazioni di seguito indicate evidenziando la presenza di eventuali overflow oppure prestiti.

$$152 + 86 =$$

$$152 + 118 =$$

$$146 - 127 =$$

$$146 - 139 =$$

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 lezione.

## ■ Prodotto

La tabella della verità a lato riporta le quattro possibili combinazioni che otteniamo dal prodotto tra due bit:

<b>x</b>	<b>b<sub>1</sub></b>	<b>b<sub>2</sub></b>	<b>prodotto</b>	
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  $5 \times 3$  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>b<sub>2</sub></b>	<b>b<sub>1</sub></b>	<b>b<sub>0</sub></b>	
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>b<sub>2</sub></b>	<b>b<sub>1</sub></b>	<b>b<sub>0</sub></b>	
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  $1011_2 \times 1010_2$ , cioè  $11_{10} \times 10_{10}$ :

Quindi il prodotto di  $1011 \times 1010$  è uguale a  $0110\ 1110_2$ , che in binario rappresenta  $110_{10}$ .

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.

Vediamo un ultimo esempio: il prodotto di  $11001010 \times 01011000$  ( $202_{10} \times 88_{10} = 17.776_{10}$ ).

Convertendo in decimale il risultato ottenuto verifichiamo che rappresenta il numero  $17.776_{10}$ .

$b_6$	$b_5$	$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	
			1	0	1	1	x
			1	0	1	0	=
<hr/>							
			0	0	0	0	+
		1	0	1	1	-	+
	0	0	0	0	-	-	+
1	0	1	1	-	-	-	=
<hr/>							
1	1	0	1	1	1	0	

$$\begin{array}{r}
 11001010 \\
 01011000 \\
 \hline
 00000000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 11001010 \\
 11001010 \\
 00000000 \\
 \hline
 10001010111000
 \end{array}$$

**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 ad esempio  $1011 \times 10$  otteniamo il risultato della tabella a fianco che è il numero di partenza con uno 0 aggiunto a destra. ►

$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	
	1	0	1	1	x
			1	0	=
<hr/>					
			0		+
	1	0	1	-	=
<hr/>					
1	0	1	1	0	

Vediamo altri due esempi:

$  \begin{array}{r}  1011011 \times \\  10 \\  \hline  0 \\  1011011 \\  \hline  10110110  \end{array}  $	$  \begin{array}{r}  11011 \times \\  10 \\  \hline  0 \\  11011 \\  \hline  110110  \end{array}  $
---	---

### 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. ►

Vediamo altri due esempi:

$  \begin{array}{r}  1011011 \quad \times \\  100 \\  \hline  00 \\  1011011-- \\  \hline  101101100  \end{array}  $	$  \begin{array}{r}  11011 \quad \times \\  100 \\  \hline  00 \\  11011-- \\  \hline  1101100  \end{array}  $
--	--

$b_5$	$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	x
1	0	1	1			
	1	0	0			=
						0 +
						0 - =
	1	0	1	1		=
						1 0 1 0 0

### 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  $n$ :

$$\begin{array}{lll}
 1011 \times 2 & n = 1 & 2^1 = 2 \quad 10110 \\
 1011 \times 4 & n = 2 & 2^2 = 4 \quad 101100 \\
 1011 \times 8 & n = 3 & 2^3 = 8 \quad 1011000
 \end{array}$$



#### 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.



### Prova adesso!

Eseguire la moltiplicazione fra i numeri binari:

$$1010 \times 101 = \quad 110101 \times 1011 = \quad 111011 \times 10111 = \quad 111111 \times 11111 =$$

- Moltiplicazione binaria

## ■ 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	0	1	=
1	1	0	1	1	0					

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 effettuiamo ottenendo:

$b_5$	$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	:	1	0	1	=
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										
<hr/>												
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										
<hr/>												
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<sub>0</sub>:

$b_5$	$b_4$	$b_3$	$b_2$	$b_1$	$b_0$					=		
1	1	0	1	1	0	:	1	0	1	=	1	0
1	0	1										
<hr/>												
0	0	1	1	1								
<hr/>												
1	0	1										
<hr/>												
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										
<hr/>												
0	0	1	1	1								
<hr/>												
1	0	1										
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									
<hr/>												
0	1	0	0									

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:	$100_2 = 4_{10}$

$$\text{quindi } 54_{10} : 5_{10} = 10_{10} \text{ con resto } 4_{10}$$

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	0	0	1
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       $110111 : 110 = 1001$

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	0	1
1	0	1	1								
			1	resto							

che è il numero di partenza senza l'ultima cifra, che rimane come resto. Per esempio:

$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	:	1	0	=	1	1	0
1	1	0	1	1							
			1	resto							

$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	:	1	0	=	1	0	1
1	0	1	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.

### 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:

$$\begin{array}{lll} 10111011 : 2 & n = 1 & 2^1 = 2 \quad 1011101 & \text{resto } 1 \\ 10111011 : 4 & n = 2 & 2^2 = 4 \quad 101110 & \text{resto } 11 \\ 10111011 : 8 & n = 3 & 2^3 = 8 \quad 10111 & \text{resto } 011 \end{array}$$



### 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.



### Prova adesso!

- Divisione binaria

Eseguire la divisione fra i numeri binari:

$$11001 : 101 = \text{con Resto} =$$

$$11111 : 110 = \text{con Resto} =$$

$$1011011 : 1101 = \text{con Resto} =$$

$$1111011 : 10011 = \text{con Resto} =$$

## Verifichiamo le competenze

### 1. Esercizi

**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 \_\_\_\_\_



# Numeri binari relativi

In questa lezione 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**;
- ▶ [-127..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;
- ▶ [-32767..0..+32767] numeri interi con segno.

Si utilizzano varie rappresentazioni:

- ▶ modulo e segno;
- ▶ in complemento a 1;
- ▶ in complemento a 2.

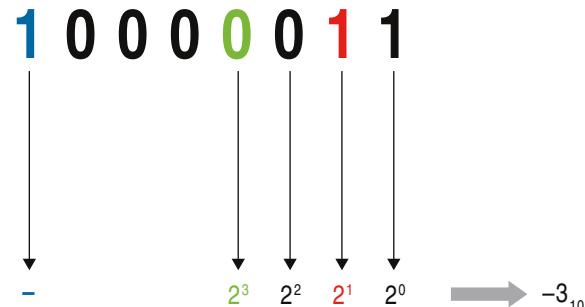
## ■ Modulo e segno

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:

$$\begin{aligned} (-2)_{10} &= (1\ 0000010)_2 \\ (+5)_{10} &= (0\ 0000101)_2 \\ (-15)_{10} &= (1\ 0001111)_2 \\ (+15)_{10} &= (0\ 0001111)_2 \end{aligned}$$



### 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). ►

Abbiamo 3 bit, quindi 8 combinazioni:

- 3 numeri positivi;
- 3 numeri negativi;
- lo 0 codificato 2 volte (+0 e -0).

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

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

$$\begin{aligned} n = 3 \quad [-2^{3-1} + 1, +2^{3-1} - 1] &= [-3, +3] \\ n = 4 \quad [-2^{4-1} + 1, +2^{4-1} - 1] &= [-7, +7] \\ n = 8 \quad [-2^{8-1} + 1, +2^{8-1} - 1] &= [-127, +127] \end{aligned}$$



## 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	1	0	1	+	
-5	=	1	0	0	0	1	0	1	=	
<hr/>										
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 lezioni 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**:

- **1**: bit di segno = numero **negativo**;
- **0101**: 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  $(01111)_2 = (+15)_{10}$ .

- **11111**:

- **1**: bit di segno = numero **negativo**;
- **1111**: valore assoluto = 15.

Il numero relativo corrispondente è quindi  $(11111)_2 = (-15)_{10}$ .

## 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 traboccamiento del **penultimo bit** (**carry-out** dal penultimo bit):

riporti	000	<b>110</b>	<b>1110</b>
	0010 +	0111	1011 +
	0100 =	0010	1110 =
	<hr/>	<hr/>	<hr/>
	0110	<b>1001</b>	<b>11001</b>
corretto		<b>overflow</b> <b>tra numeri</b> <b>positivi</b>	<b>overflow</b> <b>tra numeri</b> <b>negativi</b>

- 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

Eseguiamo  $-5 + 2 =$

Convertiamo i due operandi in modulo e segno:

$$\begin{array}{r} 1101 + \\ 0010 = \\ \hline \end{array}$$

Individuiamo il maggiore dei due operandi come valore assoluto (5).

Trasformiamo l'operando minore cambiandolo di segno:

$$-(0\ 010) \Rightarrow 1\ 010$$

Ora possiamo eseguire la somma come sottrazione:

$$\begin{array}{r} 1\ 101 - \\ 1\ 010 = \\ \hline 1\ 011 \end{array}$$

Il risultato è  $-3$ , come ci aspettavamo.

### ESEMPIO

Eseguiamo  $-5 + 7 =$

Convertiamo i due operandi in modulo e segno:

$$\begin{array}{r} 1101 + \\ 0111 = \\ \hline \end{array}$$

Individuiamo il maggiore dei due operandi come valore assoluto (7).

Trasformiamo l'operando minore cambiandolo di segno:

$$-(1\ 101) \Rightarrow 0\ 101$$

Ora possiamo eseguire la somma come sottrazione:

$$\begin{array}{r} 0\ 111\ - \\ 0\ 101\ = \\ \hline 0\ 010 \end{array}$$

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:

CA1 (5)	$\Rightarrow 5$	$= 0101$
CA1 (-5)	$\Rightarrow \text{comp}(0101)$	$= 1010$
CA1 (15)	$\Rightarrow 15$	$= 01111$
CA1 (-15)	$\Rightarrow \text{comp}(01111)$	$= 10000$
CA1 (0)	$\Rightarrow 0$	$= 00000$

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:

$$\begin{aligned} \text{CA1}(01101) &= 0 \cdot (-2^{5-1} + 1) + 1 \cdot 2^{4-1} + 1 \cdot 2^{3-1} + 0 \cdot 2^{2-1} + 1 \cdot 2^{1-1} = 0 + 2^3 + 2^2 + 0 + 2^0 = 13_{10} \\ \text{CA1}(11101) &= 1 \cdot (-2^{5-1} + 1) + 1 \cdot 2^{4-1} + 1 \cdot 2^{3-1} + 0 \cdot 2^{2-1} + 1 \cdot 2^{1-1} = (-16 + 1) + 2^3 + 2^2 + \\ &\quad + 0 + 2^0 = -2_{10} \end{aligned}$$

$$\begin{aligned} \text{CA1}(00101) &= 0 \cdot (-2^{5-1} + 1) + 0 \cdot 2^{4-1} + 1 \cdot 2^{3-1} + 0 \cdot 2^{2-1} + 1 \cdot 2^{1-1} = 0 + 0 + 2^2 + 0 + 2^0 = 5_{10} \\ \text{CA1}(10101) &= 1 \cdot (-2^{5-1} + 1) + 0 \cdot 2^{4-1} + 1 \cdot 2^{3-1} + 0 \cdot 2^{2-1} + 1 \cdot 2^{1-1} = (-16 + 1) + 0 + 2^2 + \\ &\quad + 0 + 2^0 = -10_{10} \end{aligned}$$

## 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:

$$5 = 0101$$

$$-5 = \text{CA2}(-0101)_2 = 1010_{\text{CA1}} + 0001 = 1011_{\text{CA2}}$$

- codifichiamo la seconda coppia di numeri sempre con 4 bit:

$$3 = 0011$$

$$-3 = \text{CA2}(-0011)_2 = 1100_{\text{CA1}} + 0001 = 1101_{\text{CA2}}$$

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.

### ESEMPIO

Rappresentazione in CA2 con 8 bit del numero decimale -67:

- 1 calcoliamo il valore binario di  $67_{10} = 01000011_2$ ;
- 2 trasformiamolo in CA2:  $\text{CA2}(-67_{10}) = 10111101_{\text{CA2}}$ .

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

Rappresentazione in CA2 con 8 bit del numero decimale -3.

$3_{10} = 11_2$  quindi  $-3 = 01$ , ma questo risulta essere un numero positivo!

Un errore tipico è quello di **dimenticare** il numero di bit con il quale deve essere fatta la rappresentazione.

La giusta codifica è la seguente:

- 1 calcoliamo il valore binario su 8 bit di  $3 = 00000011_2$ ;
- 2 trasformiamolo in CA2:  $-3 = 11111101_{\text{CA2}}$ .

### ESEMPIO

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	$\geq 0$	$\geq 0$	1	Sì
A + B	$< 0$	$< 0$	0	Sì
A - B	$\geq 0$	$< 0$	1	Sì
A - B	$< 0$	$\geq 0$	0	Sì

Possiamo anche formulare una regola che ci permetta 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

$$127_{10} = 01111111_{CA2}$$

riporti n e n+1 discordi overflow		segni concordi	
Riporto	0 1	1 1 1 1 1 1 1	
+127 +	0	1 1 1 1 1 1 1	+
+1 =	0	0 0 0 0 0 0 1	=
_____			
-128	1	0 0 0 0 0 0 0	
			segno risultato discorde

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:

riporti n e n+1 concordi	
Riporto	0 0
+100 +	0 1 1 0 0 1 0 0 +
+25 =	0 0 0 1 1 0 0 1 =
_____	
+125	0 1 1 1 1 0 1
	corretta
	segni concordi

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

Eseguiamo la sottrazione tra +5 e +3 con 6 bit, cioè

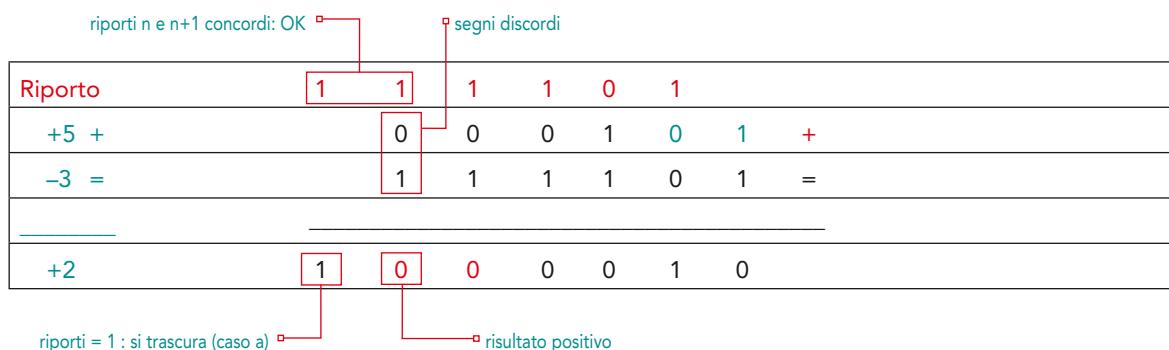
$$(+5)_{10} - (+3)_{10} = \text{cioè } (000101)_2 - (000011)_2 =$$

Innanzitutto determiniamo il **range** di rappresentazione, che è  $[-2^{6-1}, +2^{6-1}-1] = [-32, +31]$ ; ora trasformiamo l'operazione in una somma:

$$(+5)_{10} + (-3)_{10} \Rightarrow (000101)_2 + (-000011)_2$$

e codifichiamo il numero negativo in complemento a 2 ottenendo  $b_0$

$$000101_2 + 111101_{\text{CA2}} =$$



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 1, si prende il numero risultante come positivo. Ricapitolando:  $(+5)_{10} - (+3)_{10} = 000101_2 - 000011_2 = 000101_2 + 111101_{\text{CA2}} = 000010_2 = 2_{10}$ .

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  $000101_2$  e  $000101_{\text{CA2}}$  indicano lo stesso numero positivo.

### ESEMPIO

Vediamo un altro esempio, dove eseguiamo la sottrazione tra  $+13$  e  $+23$  con 6 bit, cioè

$$(13)_{10} - (23)_{10} = \text{cioè } (001101)_2 - (011010)_2 =$$

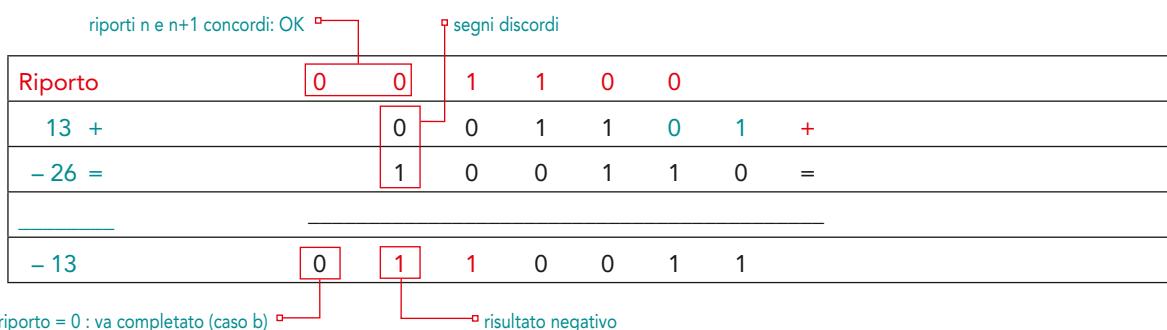
Innanzitutto determiniamo il **range** di rappresentazione, che è  $[-2^{6-1}, +2^{6-1}-1] = [-32, +31]$ ; ora trasformiamo l'operazione in una somma:

$$(13)_{10} + (-23)_{10} \Rightarrow (001101)_2 + (-011010)_2$$

Dato che ora i due addendi hanno **segno discordo**, non avremo overflow.

Codifichiamo il numero negativo in complemento a 2 ottenendo

$$001101_2 + 100110_{\text{CA2}} =$$



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.

$$\text{Ricapitolando: } (+13)_{10} + (-26)_{10} = - (110011)_{\text{CA2}} = - (001101)_2 = - 13_{10}.$$

### ESEMPIO

Vediamo un altro esempio, dove eseguiamo la sottrazione tra  $-25$  e  $+13$  con 6 bit, cioè:

$$(-25) - (+13) = \text{cioè } (100111)_{\text{CA2}} - (001101)_2 =$$

Innanzitutto determiniamo il **range** di rappresentazione, che è  $[-2^{6-1}, +2^{6-1}-1] = [-32, +31]$ ; trasformiamo l'operazione in una somma:

$$(-25) + (-13) \Rightarrow 100111_{\text{CA2}} + (-001101)_2 =$$

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

$$100111_{\text{CA2}} + 110011_{\text{CA2}} =$$

Riporto	1	0	0	1	1	1	
riporti $n$ e $n+1$ discordi: overflow							
$-25 +$	1	0	0	1	1	1	+
$-13 =$	1	1	0	0	1	1	=
---							
???	1	0	1	1	0	1	0
							risultato positivo: errore

I riporti nelle colonne  $n$  e  $n + 1$  sono diversi: siamo in presenza di overflow.

Il risultato ottenuto,  $-26$ , è errato: infatti avremmo dovuto ottenere  $-38$ , che non è contemplato nel range codificabile con 6 bit.

### ESEMPIO

Vediamo un ultimo esempio, dove eseguiamo la sottrazione tra  $-54$  e  $-44$  con 8 bit, cioè

$$(-54) + (-13) = \text{cioè } (11001010)_{\text{CA2}} - (11010100)_{\text{CA2}} =$$

Innanzitutto determiniamo il **range** di rappresentazione, che è  $[-2^{8-1}, +2^{8-1}-1] = [-128, +127]$ .

Dato che i due addendi hanno **segno uguale**, esiste la possibilità di avere overflow.

Eseguiamo la somma in complemento a 2:

Riporto	<b>1</b>	<b>1</b>	0	0	0	0	0	0	segni concordi: ctr range
- 54 +		<b>1</b>	1	0	0	<b>1</b>	0	1	
- 44 =		<b>1</b>	1	0	1	0	1	0	=
<hr/>									
- 98	<b>1</b>	<b>1</b>	0	0	1	1	1	1	0

riporti n e n+1 concordi: OK

risultato negativo

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:  $1001\ 1110_{CA2} = -01100010_2$ .

In effetti risulta un numero negativo pari a  $-01100010_2 = -98_{10}$ , che è proprio  $-54 - 44$ , ed è compreso nel range di rappresentazione =  $[-128, +127]$ .



## Prova adesso!

• Aritmetica in complemento a 2

Effettuare le seguenti addizioni segnalando eventuali overflow:

$$80 + 20 = \dots + \dots = \dots \quad CA2$$

$$90 + 30 = \dots + \dots = \dots \quad CA2$$

$$100 + 40 = \dots + \dots = \dots \quad CA2$$

Effettuare le seguenti sottrazioni con 6 bit segnalando eventuali overflow:

$$6 - 3 = \dots + \dots = \dots \quad CA2$$

$$-4 - 7 = \dots + \dots = \dots \quad CA2$$

$$-16 - 11 = \dots + \dots = \dots \quad CA2$$

## ■ 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 seconda 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 CA2 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:  $(+011)_2 \Rightarrow (1101)_{CA2} \Rightarrow (0101)_{EC8} (= -3 + 8 = 5)$ ;
- +4: 1100, così ottenuto:  $(-100)_2 \Rightarrow (0100)_{CA2} \Rightarrow (1100)_{EC8} (= +4 + 8 = 12)$ ;
- +7: 1001, così ottenuto:  $(-111)_2 \Rightarrow (0001)_{CA2} \Rightarrow (1001)_{EC8} (= +9 + 8 = 17)$ .

### 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

15	11111
14	11110
	...
2	10010
1	10001
0	10000
-1	01111
-2	01110
	...
-15	00001
-16	00000

Completiamo la codifica di tutte le possibili configurazioni, come riportato nella tabella a lato: ►

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  $15_{10}$ . 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  $2^{n-1} = 2^2 = 4$ :  
 $\text{decod}(111)_{\text{EC}4} = 7 - 4 = 3$ ;

0111 => rappresenta 7 in binario su 4 bit, quindi  $2^{n-1} = 2^3 = 8$ :  
 $\text{decod}(0111)_{\text{EC}8} = 7 - 8 = -1$ ;

1011 => rappresenta 11 in binario su 4 bit, quindi  $2^{n-1} = 2^3 = 8$ :  
 $\text{decod}(1011)_{\text{EC}8} = 11 - 8 = 3$ ;

01011 => rappresenta 11 in binario su 5 bit, quindi  $2^{n-1} = 2^4 = 16$ :  
 $\text{decod}(01011)_{\text{EC}16} = 11 - 16 = -5$ .

Questo tipo di notazione è poco usata ma è alla base della rappresentazione in virgola mobile (*floating point*) trattata nella prossima lezione.



## Prova adesso!

- Codifica in eccesso  $2^{n-1}$

Effettuare le seguenti conversioni in eccesso 8 su 4 bit con entrambe le regole pratiche precedentemente descritte:

$$2 = \dots + \dots = \dots \text{ EC8}$$

$$5 = \dots + \dots = \dots \text{ EC8}$$

$$-4 = \dots + \dots = \dots \text{ EC8}$$

Effettuare le seguenti conversioni in eccesso 16 su 5 bit con entrambe le regole pratiche precedentemente descritte:

$$2 = \dots + \dots = \dots \text{ EC16}$$

$$5 = \dots + \dots = \dots \text{ EC16}$$

$$-4 = \dots + \dots = \dots \text{ EC16}$$

## Verifichiamo le competenze

### 1. Esercizi

**1** Rappresenta in modulo e segno su 8 bit i numeri  $+37_{10}$ ,  $-37_{10}$ ,  $-100_8$ ,  $-3B_{16}$ :

$$\begin{array}{l} +37_{10} \rightarrow \underline{\hspace{2cm}} \\ -37_{10} \rightarrow \underline{\hspace{2cm}} \\ -100_8 \rightarrow \underline{\hspace{2cm}} \\ -3B_{16} \rightarrow \underline{\hspace{2cm}} \end{array}$$

**2** Converti in decimale i seguenti numeri in codice modulo e segno su 4 bit:

$$\begin{array}{l} 0011 \rightarrow \underline{\hspace{2cm}} \\ 0111 \rightarrow \underline{\hspace{2cm}} \\ 1010 \rightarrow \underline{\hspace{2cm}} \\ 1111 \rightarrow \underline{\hspace{2cm}} \end{array}$$

**3** Effettua le seguenti somme algebriche tra numeri in codice modulo e segno:

$$\begin{array}{rcl} 0010 & + & 1011 & + & 1101 & + & 1101 \\ 0100 & = & 1110 & = & 0010 & = & 0111 \\ \hline & & \hline & & \hline & & \hline \end{array}$$

**4** Dati i seguenti numeri decimali positivi, dopo averli codificati in binario su 8 bit, individua la rappresentazione del corrispondente numero negativo in modulo e segno, complemento a 1 e complemento a 2:

Binario	Modulo e segno	Complemento a 1	Complemento a 2
+18			
+115			
+79			
+69			
+3			
+100			
+126			
+111			
+99			
+64			
+10			
+56			
+16			
+55			
+42			
+121			

## 2. 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:

$$51 = \underline{\hspace{2cm}}_2$$

$$98 = \underline{\hspace{2cm}}_2$$

Rappresentazione in CA2:

$$-51 = \underline{\hspace{2cm}}_{CA2}$$

$$-98 = \underline{\hspace{2cm}}_{CA2}$$

Somma algebrica:

$$\underline{\hspace{2cm}} +$$

$$\underline{\hspace{2cm}} =$$

-----

-----

Commento

C'è overflow?

Sì [ ]      No [ ]

dato che \_\_\_\_\_

**2**  $(-54)_{10} + (-44)_{10}$

Conversione in binario dei valori assoluti:

$$54 = \underline{\hspace{2cm}}_2$$

$$44 = \underline{\hspace{2cm}}_2$$

Rappresentazione in CA2:

$$-54 = \underline{\hspace{2cm}}_{CA2}$$

$$-44 = \underline{\hspace{2cm}}_{CA2}$$

Somma algebrica:

$$\underline{\hspace{2cm}} +$$

$$\underline{\hspace{2cm}} =$$

-----

-----

Commento

C'è overflow?

Sì [ ]      No [ ]

dato che \_\_\_\_\_

**3**  $(96)_{10} + (69)_{10}$

Conversione in binario dei valori assoluti:

$$96 = \underline{\hspace{2cm}}_2$$

$$69 = \underline{\hspace{2cm}}_2$$

Rappresentazione in CA2:

$$96 = \underline{\hspace{2cm}}_{CA2}$$

$$96 = \underline{\hspace{2cm}}_{CA2}$$

Somma algebrica:

$$\underline{\hspace{2cm}} +$$

$$\underline{\hspace{2cm}} =$$

-----

-----

Commento

C'è overflow?

Sì [ ]      No [ ]

dato che \_\_\_\_\_

**4**  $(+50)_{10} + (-50)_{10}$

Conversione in binario dei valori assoluti:

$$50 = \underline{\hspace{2cm}}_2$$

$$50 = \underline{\hspace{2cm}}_2$$

Rappresentazione in CA2:

$$50 = \underline{\hspace{2cm}}_{CA2}$$

$$-50 = \underline{\hspace{2cm}}_{CA2}$$

Somma algebrica:

$$\underline{\hspace{2cm}} +$$

$$\underline{\hspace{2cm}} =$$

-----

-----

Commento

C'è overflow?

Sì [ ]      No [ ]

dato che \_\_\_\_\_

**AREA digitale**



Esercizi per il recupero / Esercizi per l'approfondimento

 [hoepliscuola.it](http://hoepliscuola.it)



# Numeri reali in virgola mobile

In questa lezione 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 *soprinsieme* 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:  $\sqrt{2}$ ,  $\pi$ , ...);
  - ▶ 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

$$\frac{1}{3} = (0.3333333333\dots)_{10} = (0.\overline{1})_3$$

$$\frac{8}{7} = (1.142857142857\dots)_{10} = (1.\overline{142857})_7$$

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 SF15-T, che approssimativamente è di 640 kg  $\sim 2^{19}$  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<sub>2</sub>  $\sim 10^{-23}$  grammi  $\sim 2^{-80}$  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:

$$N = m \cdot B^{\text{esp}} + r$$

Esistono **infinte 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 il numero **N = 31.4357**, con **B = 10** e **n = 4** cifre:

esp	scomposizione	m	r
-1	$314.3 \cdot 10^{-1} + 570 \cdot 10^{-1-4}$	314.3	.057
0	$31.43 \cdot 10^0 + 57 \cdot 10^{0-4}$	31.43	.0057
1	$3.143 \cdot 10^1 + 5.7 \cdot 10^{1-4}$	3.143	.00057
2	$.3143 \cdot 10^2 + .57 \cdot 10^{2-4}$	.3143	.000057
3	$.0314 \cdot 10^3 + .357 \cdot 10^{3-4}$	.0314	.00000357
4	$.0031 \cdot 10^4 + .4357 \cdot 10^{4-4}$	.0031	.0000004357
5	$.0003 \cdot 10^5 + .14357 \cdot 10^{5-4}$	.0003	.000000014357
6	$.0000 \cdot 10^6 + .314357 \cdot 10^{6-4}$	.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     $+45.6768 = +0.45676 \cdot 10^2$     + resto 0.0008, che è trascurabile;
- con esp = 4     $+45.6768 = +0.00456 \cdot 10^4$     + resto 0.0768, che non è trascurabile;
- con esp = 6     $+45.6768 = +0.00004 \cdot 10^6$     + resto 0.00056768, che è praticamente tutto il numero che dovevamo rappresentare;

**2** codifichiamo il numero +532.4321 e otteniamo:

- con esp = 3     $+532.4321 = +0.53243 \cdot 10^3$     + resto 0.0021, che è trascurabile;
- con esp = 5     $+532.4321 = +0.00532 \cdot 10^5$     + resto 0.4321, che non è trascurabile;
- con esp = 7     $+532.4321 = +0.00002 \cdot 10^7$     + 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:

$$N = m \cdot B^{\text{esp}}$$



### Prova adesso!

- Numeri in virgola mobile
- Numeri normalizzati

Dati i seguenti numeri reali rappresentali in formato mantissa con esponente esp = 3 e resto, indicando se tale numero è normalizzato.

numero	mantissa m	resto r	normalizzato
12.033572			sì [ ] no [ ]
612.03572			sì [ ] no [ ]
34570.172			sì [ ] no [ ]
1234.3212			sì [ ] no [ ]

## ■ La codifica binaria dei numeri reali in virgola mobile

Un numero in virgola mobile può essere scritto trascurando il resto secondo il seguente formato:

$$N = \text{mantissa} \cdot 2^{\text{esponente}}$$

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

Normalizziamo il seguente numero binario:

$$101.111_2$$

“spostiamo” la virgola a **sinistra** di 2 posizioni ottenendo il numero seguente:

$$1.01111 \cdot 2^2$$

il numero ottenuto è normalizzato.

#### ESEMPIO

Normalizziamo il seguente numero binario:

$$0.001011_2$$

“spostiamo” la virgola a **destra** di **3** posizioni ottenendo il numero seguente:

$$1.0110 \cdot 2^3$$

il numero ottenuto è normalizzato.

### ESEMPIO

Normalizziamo il seguente numero binario:

$$100100_2$$

“spostiamo” la virgola di **sinistra** di **5** posizioni ottenendo il numero seguente:

$$1.001 \cdot 2^5$$

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.



### Prova adesso!

- Codifica della mantissa normalizzata

Dati i seguenti numeri binari normalizza il valore della mantissa indicando il valore dell'e-sponente (posizioni a destra o sinistra di spostamento della virgola).

numero	mantissa normalizzata	e-sponente
111.01001		
11.010101		
0.0101101		
1.0010101		

### 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

$$N = 0.\textcolor{red}{1}xxxxxxx \cdot 2^{\text{esp}}$$

2 con bit **nascosto**

$$N = 0.\textcolor{blue}{1}\textcolor{green}{xxxxxxx} \cdot 2^{\text{esp}}$$

sottinteso

**B)** prima cifra significativa diversa da 0 alla **sinistra** del punto di radice:

1 senza bit nascosto

$$N = \textcolor{red}{1}.xxxxxxx \cdot 2^{\text{esp}}$$

2 con bit **nascosto**

$$N = \textcolor{blue}{1}.\textcolor{green}{xxxxxxx} \cdot 2^{\text{esp}}$$

sottinteso

Quella utilizzata nella rappresentazione IEEE-P754 è l'ultima.

## ■ Codifica dell'esponente

L'esponente può essere codificato in una delle forme 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”).

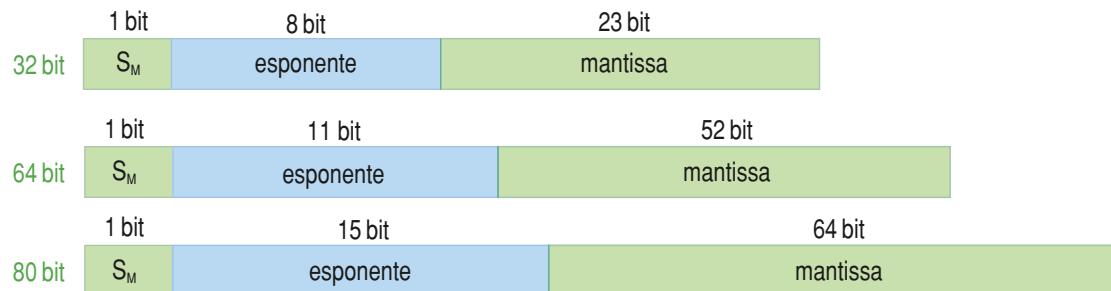
Noi vedremo la rappresentazione dell'esponente nel formato **IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)**, sinteticamente indicato come **IEEE-754**, che è lo standard più diffuso nel campo del calcolo automatico.

## ■ Rappresentazione in floating point nello standard IEEE-P754

È la rappresentazione standard utilizzata per codificare i reali nei processori della famiglia Intel 80 · 86. Sono previsti tre diversi formati:

- A** singola precisione: 32 bit;
- B** doppia precisione: 64 bit;
- C** 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.

Riportiamo uno schema riassuntivo dove possiamo confrontare i parametri delle due rappresentazioni che descriveremo in seguito nella nostra trattazione:

Elemento	Singola Precisione	Doppia Precisione
Numero di bit di segno	1	1
Numero di bit esponente	8	11
Numero di bit della frazione F	23	52
Numero di bit, totale	32	64
Rappr. esponente	Eccesso 127	Eccesso 1023
Intervallo esponente	-126 ... +127	-1022 ... +1023

La precisione della rappresentazione è stabilita dal numero di cifre della mantissa: questo valore è quindi il massimo errore (o errore relativo).

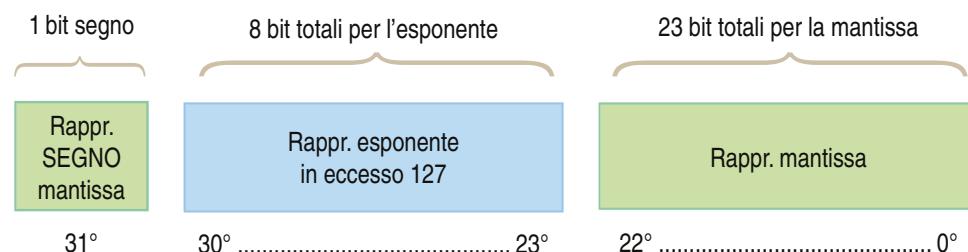
## Float in singola precisione IEEE-P754 a 32 bit

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

$N = 1.\text{xxxxxxxx} \cdot 2^{\text{esp}}$  che viene sottintesa, 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  $\text{esp}$ , 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

Codifichiamo il numero  $V = 1.0_{10}$ :

- 1 il numero è positivo, quindi il bit di segno è uguale a 0;
- 2 come primo passo codifichiamo il numero in binario:

$$1.00_2$$

- 3 procediamo con la **normalizzazione**, spostando a destra la virgola di 0 posizioni:

$$V = 1.00 \cdot 2^0 \text{ (rappresentato senza errori)}$$

- 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

Codifichiamo il numero  $V = 8.5_{10}$

- 1 il numero è positivo, quindi il bit di segno è uguale a 0 ;
- 2 come primo passo codifichiamo il numero in binario:

$$1000.1_2$$

3 ora procediamo con la **normalizzazione**, spostando a destra la virgola di 3 posizioni:

$$1.0001_2 \cdot 2^3 \text{ (rappresentato senza errori);}$$

4 codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 3 che è il valore ottenuto dalla normalizzazione:

$$3 + 127 = 130_{10} = 1000\ 0010_2$$

Quindi otteniamo:

<b>segno</b>	<b>esponente</b>	<b>mantissa normalizzata (23 bit MSB = 1 escluso)</b>
0	1000 0010	000 1 0000 0000000 00000000

a questo punto lo scomponiamo in quattro byte:

<b>byte 1</b>	<b>byte 2</b>	<b>byte 3</b>	<b>byte 4</b>
0 100 0001	0000 1000	0000 0000	0000 0000

che in esadecimale viene scritto come **0x4108 0000**.

#### ESEMPIO

Codifichiamo il numero **V = -4.5<sub>10</sub>**

- 1 il numero è negativo, quindi il bit di segno è uguale a 1;
- 2 come primo passo codifichiamo il numero in binario:

$$100.1_2$$

3 ora procediamo con la **normalizzazione**, spostando a destra la virgola di 2 posizioni:

$$1.001_2 \cdot 2^2 \text{ (rappresentato senza errori);}$$

4 codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 3 che è il valore ottenuto dalla normalizzazione:

$$2 + 127 = 129_{10} = 10000001_2$$

Quindi otteniamo:

<b>segno</b>	<b>esponente</b>	<b>mantissa normalizzata (23 bit MSB = 1 escluso)</b>
1	1000 0001	001 0000 0000000 00000000

a questo punto lo scomponiamo in quattro byte:

<b>byte 1</b>	<b>byte 2</b>	<b>byte 3</b>	<b>byte 4</b>
1100 0000	1001 0000	0000 0000	0000 0000

che in esadecimale viene scritto come **0xC090 0000**.

#### ESEMPIO

Codifichiamo il numero **V = -13.75<sub>10</sub>**:

- 1 il numero è negativo, quindi il bit di segno è uguale a 1;
- 2 come primo passo codifichiamo il numero in binario:

$$1101.11_2$$

3 ora procediamo con la **normalizzazione**, spostando a destra la virgola di 3 posizioni:

$$1.10111_2 \cdot 2^3 \text{ (rappresentato senza errori);}$$

4 codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 3 che è il valore ottenuto dalla normalizzazione:

$$3 + 127 = 130_{10} = 10000010_2$$

Quindi otteniamo:

<b>Segno</b>	<b>esponente</b>	<b>mantissa normalizzata (23 bit MSB =1escluso)</b>
1	1000 0010	101 1100 0000 0000 0000 0000

a questo punto lo scomponiamo in quattro byte:

<b>byte 1</b>	<b>byte 2</b>	<b>byte 3</b>	<b>byte 4</b>
1100 0001	0101 1100	0000 0000	0000 0000

che in esadecimale viene scritto come 0xC15C0000.

#### ESEMPIO

Codifichiamo il numero  $V = 5.875_{10}$ :

- 1 il numero è positivo, quindi il bit di segno è uguale a 0;
- 2 come primo passo codifichiamo il numero in binario:

$$101.111_2 \text{ (senza approssimazione)}$$

3 procediamo con la **normalizzazione**, spostando a destra la virgola di 2 posizioni:

$$V = 1.01111 \cdot 2^2 \text{ (rappresentato senza errori)}$$

4 codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 2, che è il valore ottenuto dalla normalizzazione:

$$2 + 127 = 129_{10} = 1000 0001_2$$

Quindi otteniamo:

<b>segno</b>	<b>esponente</b>	<b>mantissa normalizzata (23 bit MSB =1escluso)</b>
0	1000 0001	011 1100 0000 0000 0000 0000

a questo punto lo scomponiamo in quattro byte:

<b>byte 1</b>	<b>byte 2</b>	<b>byte 3</b>	<b>byte 4</b>
0100 0000	1011 1100	0000 0000	0000 0000

che in esadecimale viene scritto come 0x40BC0000.



#### Prova adesso!

• Codifica in float IEEE-P754

Dati i seguenti numeri decimali codificali in IEEE-P754 a 32 bit:

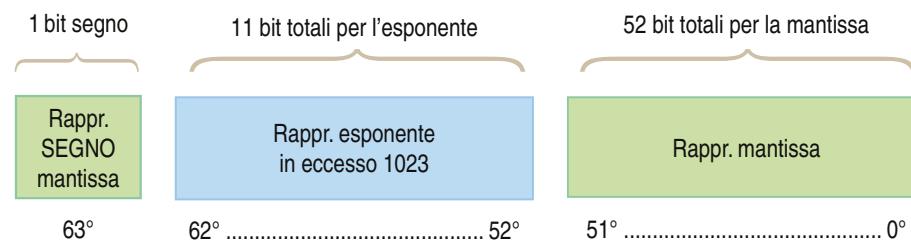
11    15.5    -5.25    -2000.125    -29.1875

## 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.

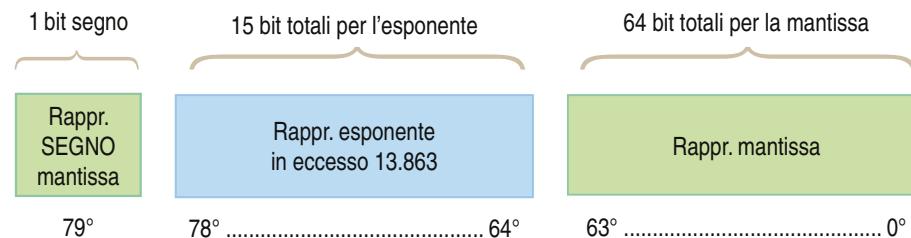


## Float in doppia precisione IEEE-P754 a 80 bit

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.



### AREA digitale



Float nel turbo C++ e Dev-Cpp

## ■ 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'**underflow** 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, 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:

## ■ Conversione da float a decimali

L'interpretazione di un numero espresso in **floating point** è piuttosto complicata. Le situazioni con **mantissa normalizzata** prevedono che l'esponente abbia un valore diverso da 0, cioè con:

$0 < \text{esponente} < 255$ : il numero è normalizzato nella forma  $(-1)^{\text{segno}} \cdot (1 + 0.\text{mantissa}) \cdot 2^{\text{esponente}}$

I casi particolari, che saranno descritti in seguito, hanno rispettivamente:

- **esponente** = 0 : mantissa denormalizzata;
  - **esponente** = 255 : codifica di  $\pm\infty$  e codifiche di errore.

Vediamo alcuni esempi con la formula

$$N = (-1)^s \cdot (1 + 0.\textcolor{teal}{mantissa}) \cdot 2^{\textcolor{red}{esponente}}$$

## ESEMPIO

Decodifichiamo il numero **V = C0400000<sub>16</sub>**:

- 1** come primo passaggio lo rappresentiamo in binario:

byte 1	byte 2	byte 3	byte 4
1100 0000	0100 0000	0000 0000	0000 0000

- 2** quindi separiamo segno, esponente e mantissa:

segno	esponente	mantissa normalizzata (23 bit MSB )
1	1000 0000	100 0000 0000 0000 0000 0000

3 il numero è negativo, dato che il bit di segno è uguale a 1;

4 il valore dell'esponente è:

$$y = 1000\ 0000_2 = 128$$

quindi otteniamo che la normalizzazione ha effettuato uno spostamento di:

$$e = 128 - 127 = 1$$

dove la mantissa è  $m = 1.1$

Il numero è quindi

$$n = 1.\underline{1}_2 \cdot 2^1$$

5 da cui si ricava la notazione binaria e il numero decimale:

$$n = 11_2 = 3_{10}$$

dove ricordiamo di aggiungere il segno negativo al risultato finale:

$$\mathbf{n = -3_{10}}$$

#### ESEMPIO

Decodifichiamo il numero  $V = 3F40\ 0000_{16}$

1 come primo passaggio lo rappresentiamo in binario:

byte 1	byte 2	byte 3	byte 4
0011 1111	0100 0000	0000 0000	0000 0000

2 quindi separiamo segno, esponente e mantissa:

segno	esponente	mantissa normalizzata (23 bit MSB )
0	011 1110	100 0000 0000 0000 0000 0000

3 il numero è negativo, dato che il bit di segno è uguale a 1;

4 il valore dell'esponente è:

$$y = 0111110_2 = 126$$

quindi otteniamo che la normalizzazione ha effettuato uno spostamento di:

$$e = 126 - 127 = -1$$

dove la mantissa è  $m = 1.1$

Si ottiene:

$$n = 1.\underline{1}_2 \cdot 2^{-1}$$

5 da cui si ricava la notazione binaria e il numero decimale:

$$n = 0.11_2 = 2^{-1} + 2^{-2} = 0.5 + 0.25 = 0.75_{10}$$

#### ESEMPIO

Decodifichiamo il numero  $V = 3F00\ 0000_{16}$

1 come primo passaggio lo rappresentiamo in binario:

byte 1	byte 2	byte 3	byte 4
0011 1111	0000 0000	0000 0000	0000 0000

2 quindi separiamo segno, esponente e mantissa:

segno	esponente	mantissa normalizzata (23 bit MSB )
0	0111 1110	000 0000 0000 0000 0000 0000

3 il numero è positivo dato che il bit di segno è uguale a 0;

4 il valore dell'esponente è:

$$y = 0111\ 1110_2 = 126$$

quindi otteniamo che la normalizzazione ha effettuato uno spostamento di:

$$e = 126 - 127 = -1 \text{ (1 a sinistra)}$$

dove la mantissa è  $m = 1.0 \cdot 2^{-1}$

Il numero è quindi

$$n = 0.10_2$$

5 da cui si ricava la notazione binaria e il numero decimale:

$$n = 0.10_2 = 0.5_{10}$$

che non deve essere cambiato di segno dato che è positivo.

### ESEMPIO

Decodifichiamo il numero  $V = BF60\ 0000$

1 come primo passaggio lo rappresentiamo in binario:

byte 1	byte 2	byte 3	byte 4
1011 1111	0110 0000	0000 0000	0000 0000

2 quindi separiamo segno, esponente e mantissa:

segno	esponente	mantissa normalizzata (23 bit MSB )
1	0111 1110	110 0000 0000 0000 0000 0000

3 il numero è negativo dato che il bit di segno è uguale a 1;

4 il valore dell'esponente è:

$$y = 0111\ 1110_2 = 126$$

quindi otteniamo che la normalizzazione ha effettuato uno spostamento di:

$$e = 126 - 127 = -1 \text{ (1 a sinistra)}$$

dove la mantissa è  $m = 1.11 \cdot 2^{-1}$

Il numero è quindi

$$n = 0.111_2$$

5 da cui si ricava la notazione binaria e il numero decimale:

$$n = 0.111_2 = 0.875_{10}$$

dove ricordiamo di aggiungere il segno negativo al risultato finale:

$$n = -0.875_{10}$$

### ESEMPIO

Decodifichiamo il numero  $V = 3EA8\ 0000_{16}$

1 come primo passaggio lo rappresentiamo in binario:

byte 1	byte 2	byte 3	byte 4
0011 1110	1010 1000	0000 0000	0000 0000

2 quindi separiamo segno, esponente e mantissa:

segno	esponente	mantissa normalizzata (23 bit MSB )
0	0111 1101	010 1000 0000 0000 0000 0000

3 il numero è positivo dato che il bit di segno è uguale a 0;

4 il valore dell'esponente è:

$$y = 0111\ 1101 = 125$$

quindi otteniamo che la normalizzazione ha effettuato uno spostamento di:

$$e = 125 - 127 = -2 \text{ (2 a sinistra)}$$

$$\text{dove la mantissa è } m = 1.0101 \cdot 2^{-2}$$

Il numero è quindi

$$n = 0.010101_2$$

5 da cui si ricava la notazione binaria e il numero decimale:

$$n = 0.010101_2 = 0.328125_{10}$$



## Prova adesso!

• Decodifica da float IEEE-P754

Dati i seguenti numeri esadecimali in IEEE-P754 a 32 bit ricava il valore decimale:

$$\mathbf{V1} = 1F40\ 0000_{16}$$

$$\mathbf{V2} = 3F23\ 0000_{16}$$

$$\mathbf{V3} = 4F41\ 1111_{16}$$

$$\mathbf{V4} = 3FF0\ F000_{16}$$

## Mantissa denormalizzata e casi particolari

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

$$1.0 \cdot 2^{-126}$$

in quanto la mantissa normalizzata deve avere valore 1 prima della virgola.

Togliendo questo vincolo il numero più piccolo sarebbe

$$0.000000\dots 1$$

cioè un numero con il valore 1 come ultimo bit, quindi:

$$1 \cdot 2^{-23}$$

Perciò il numero più piccolo rappresentabile diviene:

$$1.0 \cdot 2^{-126} \cdot 2^{-23} = 2^{-149}$$

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

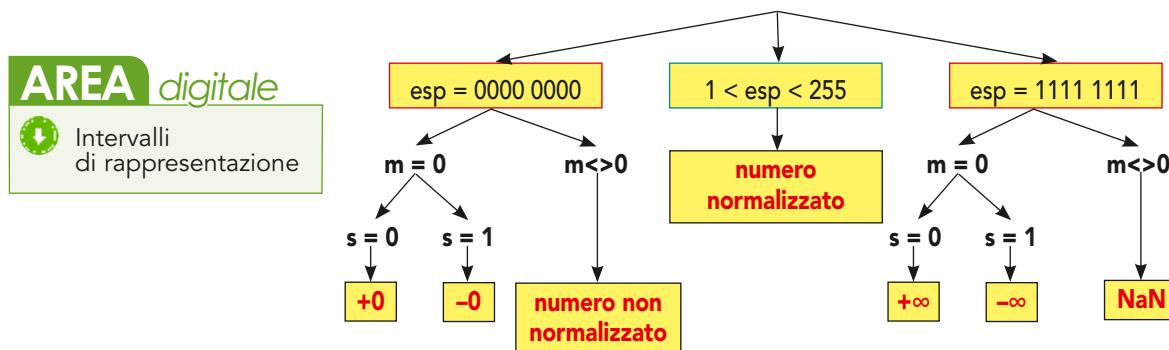
che in esadecimale viene scritto come  $0x00000001_{16}$ .

Il valore di esponente = 0000 0000 permette di codificare quindi un sottoinsieme di numeri molto piccoli, che vanno hanno come range 0 a  $2^{-149}$

Anche la codifica con l'esponente con tutti i bit a 1 (esp = 1111.1111) non viene interpretato come numero normalizzato ma permette di rappresentare:

- ▶ con  $m = 0$  e  $s = 0$ , convenzionalmente il  $+\infty$ ;
- ▶ con  $m = 0$  e  $s = 1$  e  $m = 0$ , convenzionalmente il  $-\infty$ ;
- ▶ con  $m > 0$ , un insieme di  $2^{23}$  configurazioni considerate **Not a Number (NaN)**, che sono utilizzate per applicazioni particolari.

La figura seguente schematizza tutte le rappresentazioni possibili in formato **IEEE-754**:



## ■ 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  $n$  che vogliamo rappresentare 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  $n$  che vogliamo rappresentare 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: ►

Quindi:

$$V = .1_{10} = .00011001100110011001100110011\dots_2 = \dots 0(0011)_2$$

Normalizziamo la mantissa ottenendo:

$$V = .(1100) \cdot 2^{-3} = .1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\dots \cdot 2^{-3}$$

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 all'n-sima cifra richiesta, quindi si escludono le cifre in eccesso  
 $m = .110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\dots$ ;
- **arrotondare** la mantissa aggiungendo 1 all'ultima cifra che si considera  
 $m = .11001100\ 11001100\ 11001100\dots$ .

Il C arrotonda i numeri in floating point.

### ESEMPIO

Rappresentiamo  $V = .15_{10}$ .

La **codifica binaria** è la seguente:

$$V = .15_{10} = .00(1001)_2$$

La **normalizzazione** trasforma il numero in:

$$V = (1.001) \cdot 2^{-3} = 1.001(1001) \cdot 2^{-3}$$

Con un esponente in eccesso 127 è:

$$127 - 3 = 124_{10} = 0111\ 1100_2$$

1 Con **troncamento**

segno	esponente	mantissa normalizzata (23 bit MSB )
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 0x3E199999.

## 2 Con arrotondamento

segno	esponente	mantissa normalizzata (23 bit MSB )	
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 0x3E19999A.

Per ottenere la rappresentazione con arrotondamento basta sommare 1 alla rappresentazione con troncamento.

## ESEMPIO

Rappresentiamo come secondo esempio  $V = -1/3_{10}$ .

La codifica binaria è la seguente:

$$V = -0,(3)_{10} = .(0101)_2$$

La normalizzazione trasforma il numero in:

$$V = .(0101)_2 = 1.(01)_2 \cdot 2^{-2}$$

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	0111 1101	01010101010101010101010	

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	0111 1101	0101010101010101011	

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 0xBEAAAAAA.

**ESEMPIO**

Come ultimo esempio rappresentiamo  $V = 12.6_{10}$ .

La **codifica binaria** è la seguente:

$$V = 12.6 = 1100.(1001)_2$$

La **normalizzazione** trasforma il numero in:

$$V = 1100.(1001)_2 = 1.100(1001) \cdot 2^3$$

Con un esponente in eccesso 127 è:

$$127 + 3 = 130_{10} = 10000010_2$$

**1** Con **troncamento**

segno	esponente	mantissa normalizzata (23 bit MSB )
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 )
0	10000010	1001 0011 0011 0011 0011 010

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.



**Prova adesso!**

• Codifica in float IEEE-P754

Dati i seguenti numeri decimali codificali in IEEE-P754 a 32 bit con troncamento e arrotondamento:

numero	segno	mantissa	esponente	esadecimale
-20.025				
0.3				
1.1				

## Verifichiamo le competenze

### 1. Problemi

- 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 = \_\_\_\_\_<sub>10</sub> = \_\_\_\_\_

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 = \_\_\_\_\_<sub>10</sub> = \_\_\_\_\_

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:  $.6 =$  \_\_\_\_\_<sub>2</sub>

Rappresentazione binaria:  $12.6 =$  \_\_\_\_\_<sub>2</sub>

Forma normalizzata:  $1.$  \_\_\_\_\_

Esponente: esponente = \_\_\_\_\_ quindi  $E =$  \_\_\_\_\_ + 127 = \_\_\_\_\_<sub>10</sub> = \_\_\_\_\_

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:  $E =$  \_\_\_\_\_

$\text{esp} =$  \_\_\_\_\_  $127 =$  \_\_\_\_\_

$N =$  \_\_\_\_\_ =

- 5** Ricava il valore decimale del seguente numero in virgola mobile rappresentato secondo lo standard IEEE-P754 a 32 bit:  $0\ 10000000\ 10000000000000000000000000000000$ .

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 yyyy xxxxxxxxxxxxxxxx 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.xxxxxxxxxxxxxx

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]					

**AREA** digitale



Esercizi per il recupero / Esercizi per l'approfondimento

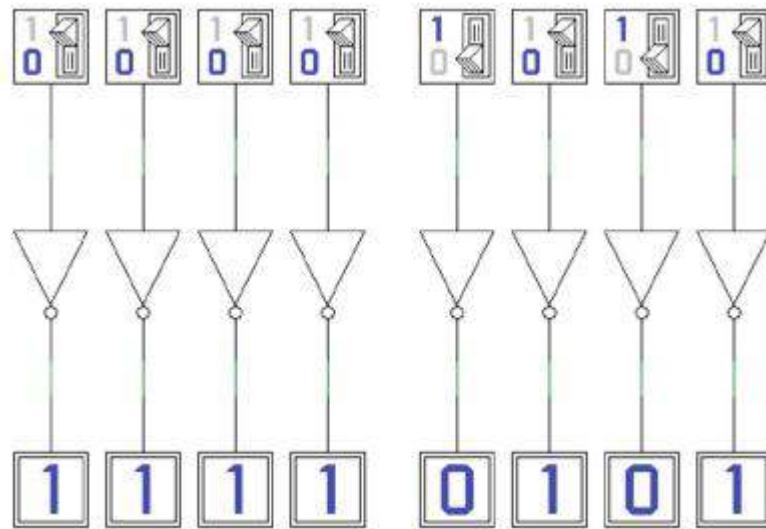
 [hoepliscuola.it](http://hoepliscuola.it)

# ESERCITAZIONI DI LABORATORIO 1

## FACCIA MO IL COMPLEMENTO A UNO E A DUE CON EXCEL

### ■ Il complemento a uno

Per rappresentare i numeri relativi in base binaria possiamo utilizzare il **complemento a uno** che è un metodo di rappresentazione alternativo al **complemento a due**. Per rappresentare i numeri binari in complemento a uno dobbiamo soltanto negare ogni bit della cifra originale.



Ad esempio per calcolare il **complemento a uno** del numero **19**, rappresentato su 8 bit (**00010011**), dobbiamo effettuare la **negazione logica bit a bit** dell'intera cifra (**11101100**). Riassumendo abbiamo che:

$$19 = 00010011$$

$$-19 = 11101100$$

Per calcolarne il suo **valore assoluto** dobbiamo nuovamente effettuare una **negazione bit a bit**.

In complemento a uno esistono due differenti rappresentazioni del numero **zero**, quella con tutti zero e quella con tutti uno:  
0000 0000 (0+)  
1111 1111 (0-)

## ■ Il complemento a due

Per rappresentare i numeri relativi in base binaria viene generalmente adottato il **complemento a due** in cui abbiamo il bit più significativo o iniziale che indica il segno della cifra rappresentata. Ne deriva che tutti i numeri che cominciano con **1** sono **negativi**, mentre tutti i numeri che cominciano con **0** sono **positivi**. Per ottenere il valore assoluto di un numero binario negativo, dobbiamo **negare** tutti i bit e aggiungere 1 al numero binario risultante.

In complemento a due, con  $n$  bit, possiamo rappresentare numeri compresi fra  $-2^{(n-1)}$  e  $+2^{(n-1)} - 1$ .

Ad esempio con 8 cifre binarie possiamo rappresentare numeri compresi tra  $-2^7$  e  $+2^7 - 1$ , cioè compresi tra -128 e +127.

Questo metodo consente di avere un'unica rappresentazione dello zero, quando tutti i bit sono zero, eliminando così la ridondanza dello zero che si verifica con la rappresentazione in modulo e segno, e di operare efficientemente addizione e sottrazione sempre avendo il primo bit a indicare il segno.

ESEMPIO Calcolare il complemento a uno e a due con Excel

Vediamo come realizzare in Excel un foglio in cui calcolare prima il **complemento a uno** e quindi il **complemento a due**. Nel foglio in questione ([UA3\\_Lab\\_1 Il complemento a uno e a due.xlsx](#)) l'utente inserisce i codici binari da codificare nelle celle **C5:R5**:

Adesso inseriamo le formule per il calcolo del **complemento a uno**. Nella cella **C7** calcoliamo il **complemento a uno** del bit presente nella cella **C5**. Per fare questo utilizziamo la funzione condizionale **=SE** nel modo seguente:

$\equiv \text{SE}(\text{D5}=""":":\text{SE}(\text{D5}=0:1:0))$

Come possiamo notare viene negato il valore binario solo se la cella contiene un valore diverso da Null. Copiamo poi la cella **C7** nelle celle **D7:R7** e otteniamo il seguente risultato:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																		
3	Complemento a uno e poi a due																	
5	numero binario																	
7	complemento a uno																	
8	aggiunta 1																	

A questo punto dobbiamo aggiungere 1 al numero ottenuto; la riga 8 contiene il numero 1 in binario e viene inserito nel modo seguente:

8	aggiunta 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
---	------------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Le due righe successive servono per effettuare l'addizione tra il numero presente nella **riga 7**, rappresentato dal valore complementato a uno e il numero 1, scritto nella **riga 8**.

9	riporti	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Numero complementato	1	0	1	0	1	0	1	1	1	1	1	0	0	1	1	1

Vediamo come calcolare l'eventuale riporto, iniziando dal bit meno significativo. Nella cella **R9** assegniamo zero come primo riporto. Nella cella **Q9** calcoliamo il riporto nel modo seguente:

```
=SE(0(E(R7=1;R8=1;R9=0);E(R7=1;R8=0;R9=1);E(R7=0;R8=1;R9=1));1;0)
```

Possiamo leggere la funzione condizionale nel modo seguente:

- ▶ mettiamo 1 come riporto se dei tre bit che vengono sommati, rappresentati dal numero, dall'aggiunta e dal riporto (**R7**, **R8**, **R9**), almeno due valgono 1. In caso contrario il riporto è 0;
- ▶ copiamo quindi la cella **Q9** nelle celle adiacenti (**C9:P9**).

Infine passiamo a calcolare il risultato dell'addizione, partendo dal bit meno significativo (cella **R10**). In questo caso il calcolo è abbastanza semplice, sapendo che il bit da sommare è sempre 1 dobbiamo semplicemente verificare il contenuto della cella **R7**, se essa vale 1 il risultato è **zero** o viceversa:

```
=SE(R7="";"";SE(R7=0;1;0))
```

Nella cella **Q10** dobbiamo calcolare il risultato dell'addizione tra il numero presente nella **riga 7**, l'aggiunta di 1 (**riga 8**) e l'eventuale riporto (**riga 9**):

```
SE(0(E(Q7=1;Q8=1;Q9=0);E(Q7=1;Q8=0;Q9=1);E(Q7=0;Q8=1;Q9=1));0;SE(0(E(Q7=1;Q8=1;Q9=1);E(Q7=1;Q8=0;Q9=0);E(Q7=0;Q8=1;Q9=0);E(Q7=0;Q8=0;Q9=1));1;0))
```

La funzione condizionale che utilizziamo è piuttosto complessa, possiamo riassumerla nel modo seguente:

Se la combinazione dei tre bit della colonna precedente è la seguente:

Numero	Aggiunta	Riporto
1	1	1
1	0	1
0	1	1

Il risultato è **0**.

Altrimenti se la combinazione dei tre bit è la seguente:

Numero	Aggiunta	Riporto
1	1	1
1	0	0
0	1	0
0	0	1

Il risultato è **1**.

Altrimenti **0**.

Il foglio di lavoro nel suo complesso è il seguente:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2																			
<b>Complemento a uno e poi a due</b>																			
5	<b>numero binario</b>	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	1	=	
7	<b>complemento a uno</b>	1	0	1	0	1	0	1	1	1	1	1	1	0	0	1	1	0	
8	<b>aggiunta 1</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
9	<b>riporti</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	<b>Numero complementato</b>	1	0	1	0	1	0	1	1	1	1	1	1	0	0	1	1	1	



## Prova adesso!

- Utilizzare Excel per calcolare il complemento a uno e a due



**APRI IL FILE** UA3\_Lab\_1 Il complemento a uno e a due

- Nell'esempio proposto l'utente inserisce manualmente i valori binari dei numeri da convertire. Modifica il foglio di lavoro in modo tale che, dopo aver ricevuto il numero in base decimale da parte dell'utente, il foglio di Excel lo converta in binario.
- Effettua la stessa operazione anche per il risultato, mostrando il numero binario risultante in base decimale.

# ESERCITAZIONI DI LABORATORIO 2

## L'ADDIZIONE E LA SOTTRAZIONE IN BASE BINARIA

### ■ L'addizione binaria

L'**addizione binaria** segue le stesse regole dell'addizione matematica, pertanto dobbiamo eseguire la somma tra ciascuna cifra (bit) di un addendo con quella dell'altro numero binario in base alla posizione, con un riporto di 1 in avanti ogni qualvolta si oltrepassa il valore 1. La tabella di riferimento per l'addizione binaria è la seguente:

0 + 0	= 0
0 + 1	= 1
1 + 0	= 1
1 + 1	= 0 con riporto 1
1 + 1 + 1	= 1 con riporto 1

In generale se si sommano un numero pari di bit a 1, il risultato è 0 con riporto 1, mentre se il numero di bit a 1 da sommare è dispari il risultato è 1 con riporto 1.

### ■ La sottrazione binaria

La sottrazione di due numeri binari viene calcolata adattando le stesse regole della sottrazione dei numeri decimali. Come avviene nel sistema decimale, quando una cifra del minuendo (il numero di partenza) è minore della cifra corrispondente nel sottraendo (il numero da sottrarre), si prende a prestito una unità dalla cifra precedente (a sinistra), che così si somma al minuendo con il valore della base di numerazione. La tabella di riferimento per la sottrazione binaria è la seguente:

$0 - 0$	= 0
$0 - 1$	= 1 con prestito dalla prima colonna a sinistra
$1 - 0$	= 1
$1 - 1$	= 0

Quando si ha un prestito si sottrae 1 dalla colonna di sinistra (quella più significativa) e si procede rispettando la regola della differenza. Se sulla colonna di sinistra non si può concedere il prestito perché la cifra è 0, esso si trascina alla colonna successiva verso sinistra finché non si restituisce il prestito.

La sottrazione può anche essere effettuata come addizione tra un numero binario e un numero binario espresso in complemento a due.

ESEMPIO **Calcolare addizione e sottrazione con Excel**

Vediamo come realizzare in Excel un foglio in cui calcolare prima il **l'addizione binaria** e quindi la sottrazione binaria effettuando il **complemento a due** del secondo numero. Nel foglio in questione ([UA3\\_Lab\\_2 operazioni binarie.xlsx](#)) l'utente inserisce i codici binari dei due numeri da sommare nelle celle **D5:S5** e del secondo numero nelle celle **D7:S7**:

Nella **riga 8** viene calcolato il **riporto**, mentre nella **riga 9** il **risultato finale** dell'addizione, come possiamo vedere dalla seguente figura:

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
8	riporti	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	
9	Risultato	1	1	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	

Vediamo come calcolare il riporto: innanzi tutto il primo riporto sarà sempre **zero** (cella **S8**). Vediamo come calcolare il riporto tra i primi due bit (cella **R8**):

=SE(E(S5=1;S7=1;S8=1);1;SE(E(S5=1;S7=1;S8=0);1;SE(E(S5=0;S7=1;S8=1);1;SE(E(S5=1;S7=0;S8=1);1;0))))

La funzione condizionale che utilizziamo è piuttosto complessa, possiamo riassumerla nel modo seguente:

Se la combinazione dei bit della colonna precedente è la seguente:

Primo addendo	Secondo addendo	Riporto
1	1	1

Il risultato è 1

Altrimenti, se la combinazione dei bit della colonna precedente è la seguente:

Primo addendo	Secondo addendo	Riporto
1	1	0

Il risultato è 1

Altrimenti, se la combinazione dei bit della colonna precedente è la seguente:

Primo addendo	Secondo addendo	Riporto
0	1	1

Il risultato è 1

Altrimenti, se la combinazione dei bit della colonna precedente è la seguente:

Primo addendo	Secondo addendo	Riporto
1	0	1

Il risultato è 1

Altrimenti 0

Dopo aver copiato la cella R8 nella zona D8:Q8, calcoliamo il risultato della prima colonna nella cella S9, mediante la funzione condizionale seguente:

```
=SE(E(S5=0;S7=0);0;SE(E(S5=1;S7=1);0;1))
```

La funzione pone 0 come risultato quando entrambi i bit dei due addendi sono posti a zero, oppure a 1, mentre invece il risultato è 1 negli altri casi.

Vediamo come calcolare l'altro risultato nella cella R9:

```
SE(O(E(R5=0;R7=0;R8=0);E(R5=1;R7=1;R8=0);E(R5=1;R7=0;R8=1);E(R5=0;R7=1;R8=1));0;1)
```

Il risultato dell'addizione vale 0 quando almeno uno dei quattro casi esposti di seguito è verificato:

Primo addendo	0	1	1	0
Secondo addendo	0	1	0	1
Riporto	0	0	1	1

In tutte le altre combinazioni il risultato è 1.

Vediamo l'esecuzione di quanto enunciato:

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1																			
<b>Somma tra numeri binari</b>																			
5	<b>primo numero (max 16 bit)</b>	1	0	0	0	0	1	1	1	1	0	1	0	0	0	1			
7	<b>secondo numero (max 16 bit)</b>	0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1	=	
8	<b>riporti</b>	0	0	0	0	1	1	1	0	0	1	1	1	1	1	1			
9	<b>Risultato</b>	1	1	0	0	1	0	1	1	0	1	1	0	0	0	0			

La seconda parte del foglio effettua la sottrazione tra due cifre binarie. L'utente immette i due numeri nelle righe 15 e 17, quindi il foglio provvede a effettuare il complemento a due del secondo numero e ad addizionarlo al primo utilizzando quanto illustrato sopra:

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
11																			
<b>Sottrazione tra numeri binari con complemento a due</b>																			
15	<b>primo numero (16 bit)</b>	0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	-		
17	<b>secondo numero (16 bit)</b>	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	1	=	
19	<b>complemento a uno</b>	1	0	1	0	1	0	1	1	1	1	1	0	0	1	1	0		
20	<b>aggiunta 1</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
21	<b>riporti</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
22	<b>complemento a due</b>	1	0	1	0	1	0	1	1	1	1	1	0	0	1	1	1		
25	<b>riporti della somma</b>	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0		
26	<b>Risultato</b>	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1		



## Prova adesso!



**APRI IL FILE** UA3\_Lab\_2 operazioni binarie

- Utilizzare Excel per calcolare la somma tra due sequenze di bit
- Utilizzare Excel per calcolare la differenza tra due sequenze di bit

Nell'esempio proposto l'utente inserisce manualmente i valori binari dei numeri da sommare o sottrarre. Modifica il foglio di lavoro in modo tale che, dopo aver ricevuto il numero in base decimale da parte dell'utente, il foglio di Excel lo converta in binario.

2. Effettua la stessa operazione anche per il risultato, mostrando il numero binario risultante in base decimale, sia per l'addizione che per la sottrazione.

# 4

# Il sistema operativo

- L1 Generalità sui sistemi operativi
- ⬇️ L2 Evoluzione dei sistemi operativi
- L3 La gestione del processore
- L4 La gestione della memoria
- L5 La memoria secondaria: il file system
- ⬇️ L6 Struttura e realizzazione di un file system
- ⬇️ L7 La sicurezza del file system
- ⬇️ L8 La gestione della I/O

## Esercitazioni di laboratorio

- ① La shell dei comandi di Windows; ② I comandi principali;  
 ③ I caratteri jolly, reindirizzamento e pipelining; ④ I file batch

### Conoscenze

- 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
- I sistemi di protezione dei dati
- Conoscere l'hardware dei dispositivi di I/O

### Competenze

- Classificare i sistemi operativi
- Descrivere il ciclo di vita di un processo
- Classificare le memorie
- Riconoscere il modello client-server
- Classificare le tecniche di gestione delle periferiche

### Abilità

- Utilizzare in modo appropriato la terminologia tecnica
- Riconoscere le caratteristiche principali del sistema operativo
- Scegliere le politiche di allocazione del processore
- Individuare le problematiche per la cooperazione tra processi
- Utilizzare le tecniche di back-up dei dati

## AREA *digitale*

-  Esercizi
-  Immagini
-  Process control block
- ▶ Cambio di contesto
- ▶ Esempio di schedulazione di un progetto con Gantt
- ▶ Highest Response Ratio Next Scheduling
- ▶ Periodo dell'RTC
- ▶ Il primo calcolatore con disco magnetico
- ▶ Condivisione di file
- ▶ Accesso indirizzato
- ▶ Flag dei diritti nei file Unix
- ▶ Correggere lo stato dei supporti di memoria
- ▶ Creare un'Unità virtuale
- ▶ Esempi di comandi CD, MD e RD
- ▶ La gestione delle utenze



Soluzioni (prova adesso, esercizi, verifiche)  
 Puoi scaricare il file anche da 



# Generalità sui sistemi operativi

In questa lezione 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 un programma deve essere presente in memoria centrale per essere eseguito) 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** (**R**ead **O**nly **M**emory), 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 carichano la prima istruzione dall’indirizzo esadecimale 0xFFFFFFF0 di quello che si chiama **IPL**, **I**nitial **P**rogram **L**oader) che assume il controllo della **CPU** all’accensione;
- 3** ogni componente hardware, appena viene alimentato, manda in esecuzione un programma di autodiagnostica (**POST**, **P**ower **O**n **S**elf **T**est), 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 Münchhausen 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.



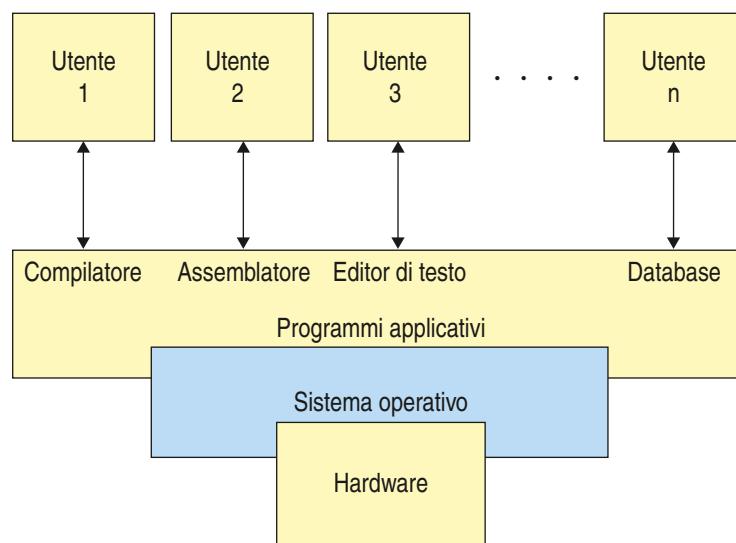
### SO (SISTEMA OPERATIVO)

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).



◀ Le **risorse** sono gli elementi **hardware** o **software** del PC che vengono usate da specifici programmi per eseguire il proprio compito. ►

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.

Il **SO** risiede sull'hard disk come tutti gli altri programmi e viene caricato nella memoria **RAM** all'accensione della macchina, o meglio, solo una sua parte viene caricata in memoria centrale: il **nucleo**.

Il **SO** è formato da un insieme di programmi organizzati tra loro in modo tale che ciascuno di essi si occupi di un compito specifico, secondo uno schema detto "a buccia di cipolla" (dall'inglese **onion skin**), elaborato da **H.M. Deitel** nel 1983.

I programmi che occupano una posizione più interna interagiscono maggiormente con l'hardware, mentre i programmi collocati più all'esterno interagiscono maggiormente con l'utente. La struttura **onion skin** mostra i programmi secondo una struttura gerarchica nella quale ciascuno strato si serve dello strato inferiore. I sei livelli sono sintetizzati come segue:

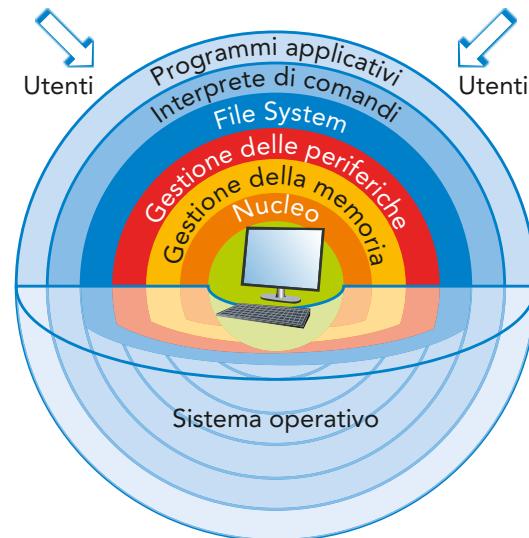
- 1° livello: **nucleo** (**kernel**)
- 2° livello: **gestore della memoria** centrale
- 3° livello: **gestore delle periferiche**
- 4° livello: **file system**
- 5° livello: **interfaccia con l'utente** (**shell**)
- 6° livello: **programmi applicativi**

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 "guiscio"), in quanto ha la funzione di fare da interfaccia tra l'utilizzatore e il nucleo, isolandolo e "proteggendolo".

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. ►

## AREA digitale

- ▶ Memoria
- ▶ Memoria virtuale



**Zoom su...**

### VIRTUALE

Nei sistemi operativi il termine "virtuale" viene utilizzato spesso: si parla di macchina virtuale, di memoria virtuale, di risorsa virtuale ecc.

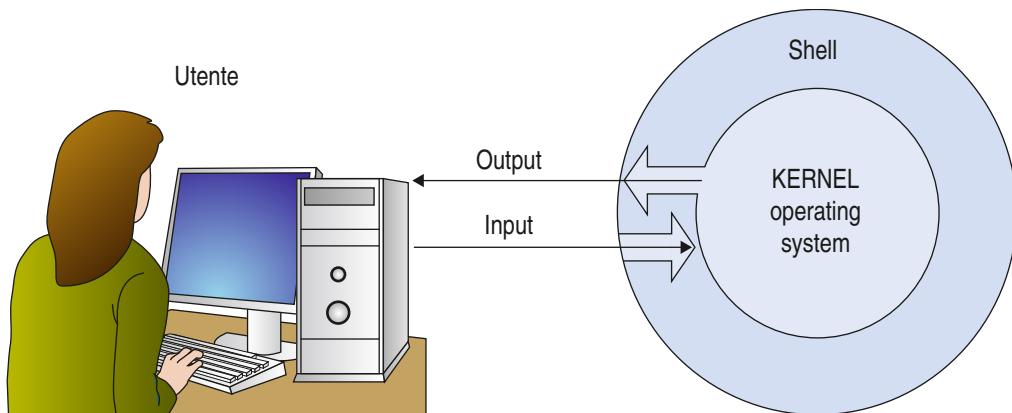
"Virtuale" sta a indicare un qualcosa che fisicamente non esiste ma viene "simulato" tramite software e all'utente finale "sembra" che sia presente realmente.

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 lezione:

- ▶ avvio e terminazione dei programmi (◀ **processi** ▶);
- ▶ assegnazione della **CPU** ai diversi **processi**;
- ▶ sincronizzazione tra i **processi**;
- ▶ sincronizzazione dei processi con l’ambiente esterno.



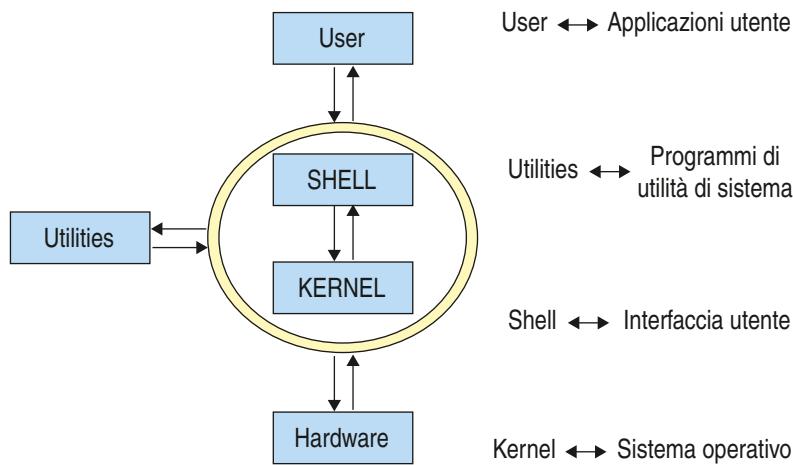
◀ **Processo** È l’insieme delle azioni compiute dal processore per eseguire un programma. Si può affermare che un **processo** è un **programma in esecuzione**. ►

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**. 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).



◀ Con **interfaccia utente** (o **shell**) si intende ciò che si frappone tra la macchina e l'utente, cioè 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).

Una finestra del terminale di Linux Red Hat. La barra superiore mostra il percorso "D:\Documents and Settings\jedjim". La finestra contiene una tabella di file e cartelle con le seguenti righe:

Modified	Time	T	Size	Perm.	Name
09/15/2000	18:37:29	d		8 B	...
09/12/2000	02:02:08	d		8 B	...
09/15/2000	01:00:08	d		8 B	...
09/15/2000	01:52:29	d		8 B	...
09/09/2000		d		8 B	...
09/06/2000	01:17:28	d		8 B	...
09/28/2000	12:07:08	n		82 B	...
09/04/2000	01:32:08	n		8 B	...
09/15/2000	01:00:08	n		17 B	...
11/21/1995	01:15:29	n		17 B	...
09/28/2000	00:54:08	n		17 B	...
09/15/2000	01:23:29	n		55 B	...
09/23/1995	01:14:29	n		16 B	...
12/23/1995	01:14:08	n		17 B	...
09/09/2000	01:04:08	n		17 B	...
09/11/2000	01:57:08	n		17 B	...
09/21/2000	01:04:08	n		8 B	...
09/15/2000	01:07:29	t		18 B	...
09/25/2000	01:00:08	n		8 B	...
09/15/2000	18:37:29	t		18 B	...
09/17/1995	04:15:29	t		2 B	...
09/15/2000	18:04:08	n		8 B	...
09/28/2000	12:07:08	n		2 B	...
09/11/2000	01:51:08	n		8 B	...
09/28/2000	00:56:08	n		2 B	...
09/09/2000	01:32:29	t		1 B	...

26 files, 8 directories

Interfaccia a linea di comando (Linux Red Hat)



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 lezioni 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 idoneo avviene tenendo conto delle esigenze dell'utente e del campo di applicazione. Alcuni sono infatti idonei a gestire reti di computer, altri invece sono più adatti 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**, per la grafica, **MacOSX** (vedi figura 2) e infine **Windows 8** (vedi figura 3) e **Android** (vedi figura 4) per la gestione dei sistemi touch.



Figura 1  
Immagine raffigurativa  
del SO Linux



Figura 2  
Immagine raffigurativa  
del SO MacOS



Figura 3  
Immagine raffigurativa  
del SO Windows



Figura 4  
Immagine raffigurativa  
del SO Android

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.

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 sistema operativo.



◀ La **versione** è molto spesso rappresentata da un numero che accompagna il nome di un programma per identificare l'eventuale aggiornamento. Il numero può andare da 1 a 10, oppure indicare l'anno di uscita del programma. Windows, a esempio, è stato prodotto nelle seguenti versioni: Windows 95 (1990); Windows 98 (1998); Windows 2000 (2000); Windows XP (2003); Windows Vista (2006); Windows 7 (2010); Windows 8 (2012); Windows 8.1 (2013). ▶

## Verifichiamo le conoscenze



### 1. Risposta multipla

**1 Il kernel di un sistema operativo:**

- a. è scritto in assembler;
- b. contiene il programma di bootstrap
- c. è caricato dopo il bootstrap
- d. è caricato prima del bootstrap

**2 L'acronimo POST significa:**

- a. Power Off Self Test
- b. Power On Security Test
- c. PC On Self Test
- d. Power On Self Test

**3 Con firmware si intende (indicare l'affermazione errata):**

- a. software non modificabile
- b. software contenuto nel BIOS
- c. componenti hardware come il BIOS
- d. software a corredo dell'HW

**4 L'acronimo BIOS significa:**

- a. Binary Input-Output System
- b. Basic Input-Output System
- c. Binary Input-Output Software
- d. Basic Input-Output Software

**5 Le primitive possono essere:**

- a. invocate esplicitamente dai processi
- b. invocate da istruzioni macchina generate dai compilatori nella traduzione di costrutti di alto livello
- c. invocate all'interno di funzioni di libreria
- d. tutte le situazioni precedenti

**6 Quale di queste affermazioni in merito ai SO è vera?**

- a. il software di base fa parte del S.O.
- b. sono scritti in binario
- c. sono residenti su disco
- d. è contenuto nella ROM

**7 Nel software di base è compreso:**

- a. sistema operativo
- b. editor
- c. BIOS
- d. traduttori
- e. linker
- f. caricatori
- g. debugger

**8 Quale di queste affermazioni in merito allo shell è vera?**

- a. fa parte del kernel
- b. viene caricata con il BIOS
- c. l'interfaccia utente fa parte dello shell
- d. è residente in ROM

**9 Quali tra i seguenti sono sistemi operativi? (3 risposte)**

- a. Bootstrap
- b. Word
- c. Excel
- d. Access
- e. Unix
- f. DOS
- g. Linux
- h. BIOS

**10 Che cosa significa la sigla GUI?**

- a. indica un programma grafico
- b. Graphical User Internet
- c. Graphical User Interface
- d. indica un sistema operativo a riga di comando

# La gestione del processore

In questa lezione 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 lezione 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**.

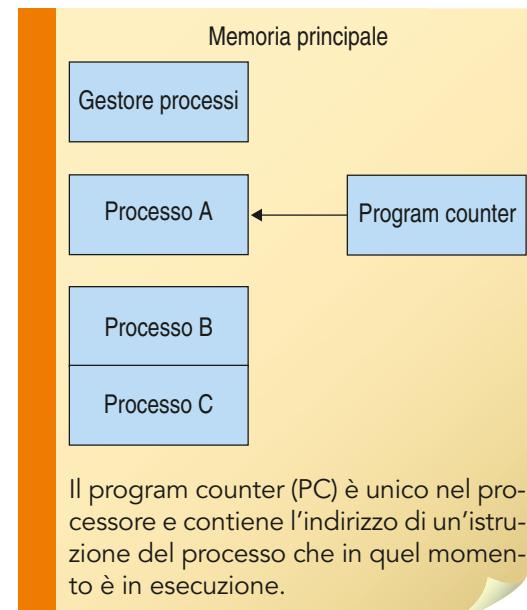
Noi prenderemo il singolo **processo** come unità di riferimento nei moderni sistemi operativi, che possiamo dire essere costituiti da due parti:

- 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 **descrittore 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:

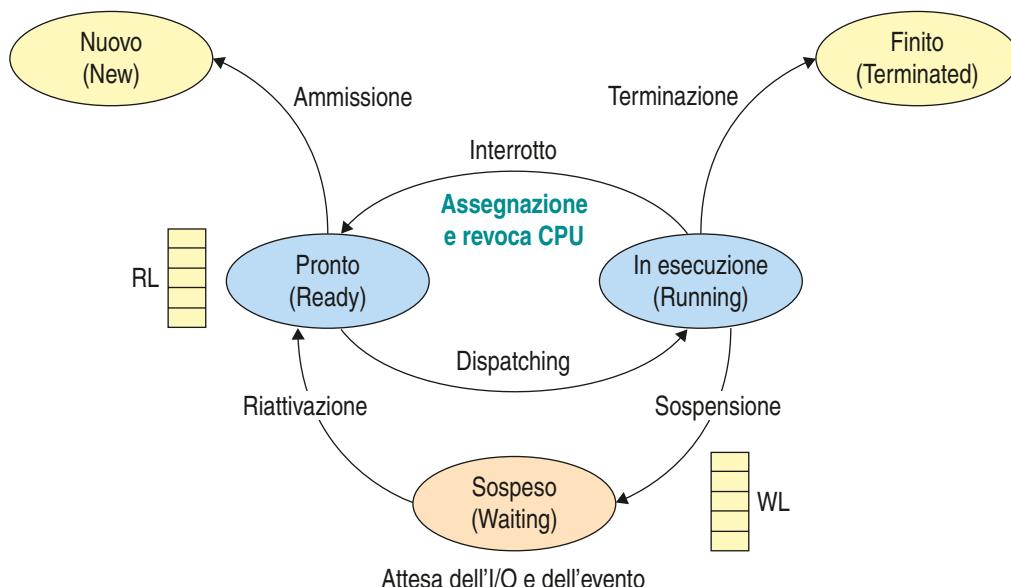
- **nuovo (new)**: è lo stato in cui si trova un processo appena è stato creato, cioè l'utente richiede l'esecuzione di un programma che risiede sul disco;
- **esecuzione (running)**: il processo sta evolvendo, nel senso che la **CPU** sta eseguendo le sue istruzioni, e quindi a esso è assegnato il processore. Nei sistemi a monoprocesso (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 assegna);
- **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*);
  - **gli manca la disponibilità di una risorsa**: per poter evolvere necessita di una risorsa che al momento non è disponibile e quindi il processo si **sospende** (*suspend*) e passa nello **stato di attesa**, insieme ad altri processi, formando la **waiting list** (◀ WL ▶).



◀ Con **RL** si indica la **ready list**, cioè la lista dei processi che si trovano nello stato di **pronto** (**ready**) e stanno aspettando il loro turno di utilizzo della **CPU**. Con **WL** si indica la **waiting list**, cioè la lista dei processi che si trovano nello stato di **attesa** (**wait**), aspettando che si liberi una risorsa oltre alla **CPU** che gli permetta di evolvere (stampante, memoria, ecc). ►

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.

### AREA digitale

Process control block

## ■ La schedulazione dei processi

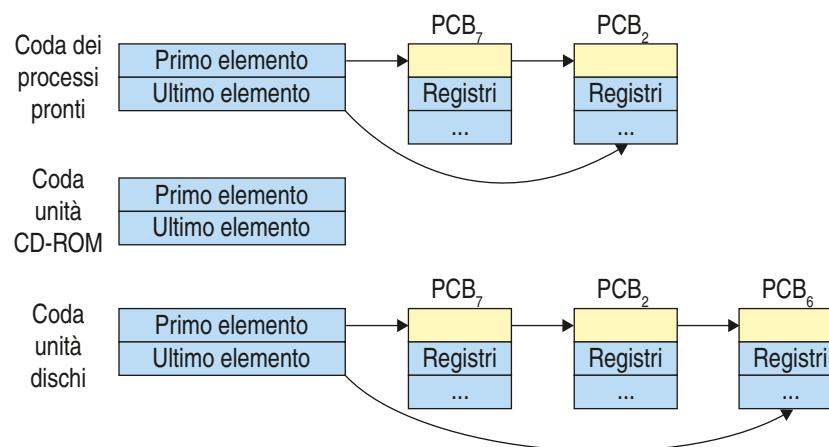
Tra le operazioni che il sistema operativo deve effettuare rientra l'assegnazione della **CPU** ai processi che sono nella ready list, cioè nella lista dei processi pronti.

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 millisecondo.

### AREA digitale



Cambio di contesto



#### 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: questi processi sono quindi **non pre-emptive**.

## ■ I criteri di scheduling

Prima di poter analizzare i criteri di scheduling è necessario introdurre la definizione di alcuni termini:



### 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.

### BURST DI CPU

Uso continuativo della CPU da parte di un processo

### BURST DI I/O

Uso continuativo di un dispositivo di I/O da parte di un processo

Gli *obiettivi primari* delle **politiche di scheduling** sono:

- ▶ massimizzare la percentuale di utilizzo della **CPU**: l'ideale sarebbe raggiungere una percentuale di utilizzo del 100% eseguendo lavoro utile, cioè “sprecare” il minor tempo per il cambio di contesto;
- ▶ massimizzare il **throughput del sistema**, cioè il numero di processi completati nell'unità di tempo;
- ▶ 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.

Possiamo inoltre elencare gli *obiettivi generali* che sono caratteristici di tutti i sistemi operativi:

- ▶ **equità**: dare a ogni processo una porzione equa della **CPU**;
- ▶ **bilanciamento**: tenere occupate tutte le parti del sistema;
- ▶ attuare politiche di **controllo**: verificare che le politiche vengano messe in atto;
- ▶ uso della **CPU**: tenere sempre occupata la **CPU**;

e *obiettivi specifici* a seconda del tipo di sistema operativo:

- ▶ **sistemi batch**
  - massimizzare il **throughput**, cioè il numero di job per unità di tempo;
  - minimizzare il **turnaround time**, cioè il tempo medio di esecuzione;
- ▶ **sistemi interattivi**
  - minimizzare i tempi di risposta (*response time*) dando l'impressione di “esclusività” tra utente e sistema;
- ▶ **sistemi real time**
  - **deadline**: rispettare le scadenze in quanto il ritardo potrebbe provocare danni irreversibili.

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

### Calcolo del tempo di attesa medio

Vediamo un esempio di come calcolare il **tempo di attesa medio** nel caso di tre processi e verifichiamo numericamente come tale risultato sia casuale e quindi non significativo da essere preso come *parametro di qualità* dato che dipendente **solo dall’ordine di arrivo dei processi stessi**.

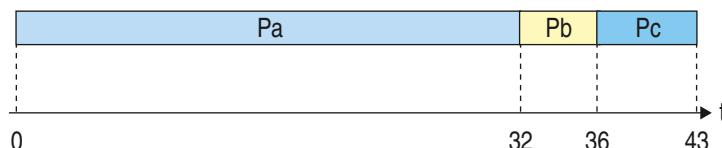
Supponiamo di avere tre processi che rispettivamente richiedono:

$P_a = 32$  unità di tempo/CPU

$P_b = 4$  unità di tempo/CPU

$P_c = 7$  unità di tempo/CPU

Nel primo caso i tre processi arrivano nell’ordine  $P_a$ ,  $P_b$  e  $P_c$ . Calcoliamo il tempo medio di attesa in questa situazione rappresentandolo su di un diagramma di **Gantt**:



$$ta_{m1} = \frac{ta_a + ta_b + ta_c}{3} = \frac{0 + 32 + 36}{3} = 22.7$$



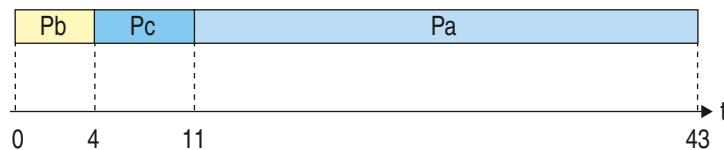
Il **Gantt** (o diagramma a barre) è uno strumento puramente grafico in cui la durata delle attività è indicata con linee e i momenti critici del progetto (**milestones**) sono indicati con simboli (per esempio dei triangoli). Il **Gantt** fornisce una visione rapida del piano temporale di un processo e/o di un progetto.

### AREA digitale



Esempio di schedulazione di un progetto con Gantt

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 sia in grado di effettuare una **stima dei tempi di utilizzo della CPU (burst di 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

#### Calcolo del tempo di attesa nel caso SJF

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:

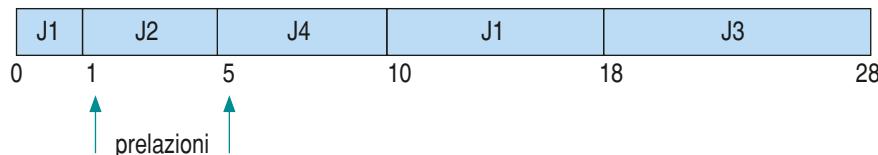
J1	J2	J4	J3
0	9	13	18

Il tempo di attesa è così calcolato, sottraendo a ogni job il “tempo di arrivo in coda”:

$$Tw = (J1, J2, J4, J3) = [0 + (9 - 1) + (13 - 3) + (18 - 2)] / 4 = 8.5$$

In questo caso si è ipotizzata una situazione di **SJF non pre-emptive**: vediamo ora come si comporterebbe un **SFRT pre-emptive**: se arriva un nuovo processo con **burst** di esecuzione più corto questo effettua una “prelazione” rispetto al processo in esecuzione.

Il job J1 va in esecuzione ma all'istante di tempo 1 giunge il processo J2 che ha un **burst** minore del **burst** rimanente di J1 (4 contro 9-1): J1 in esecuzione viene sospeso e viene mandato in esecuzione J2; analogà situazione avviene all'arrivo del job J4 che ha un **burst** di durata=5, inferiore sia a quello di J3 che del residuo di J1. Il diagramma di **Gantt** completo è il seguente:



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.



Zoom su...

### IL VALORE 0 INDICA ALTA OPPURE BASSA PRIORITÀ?

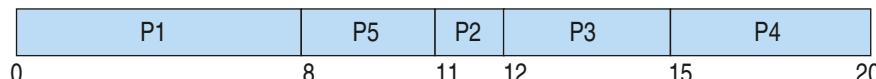
A seconda dei sistemi operativi si assegnano numeri interi con diversi significati di priorità: possiamo avere sistemi che con  $p = 0$  si associa la priorità più alta oppure alta oppure sistemi come Windows e Linux che associano a  $p = 0$  la priorità più bassa.

### ESEMPIO Sequenza di attivazione

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	3	4
P4	5	5
P5	3	2

La sequenza di attivazione è la seguente:



Il tempo di attesa medio è:

$$Tw = (P1, P5, P2, P3, P4) = (0 + 8 + 11 + 12 + 15) / 5 = 9.2$$

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 sopravvengono 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 della **CPU**.

La soluzione ottimale la si ottiene cambiando radicalmente politica, cioè adottando la **Round Robin**.

#### AREA digitale



Highest Response Ratio Next Scheduling

### 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**) di durata che varia tra 10 e -100 millisecondi.

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**.

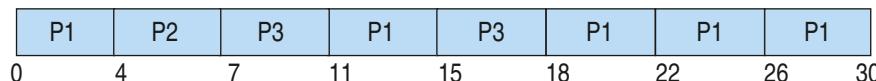
Con questo algoritmo tutti i processi sono trattati allo stesso modo, in una sorta di "correttezza" (**fairness**), e possiamo essere certi che non ci sono possibilità di **starvation** perché tutti a turno hanno diritto a utilizzare la **CPU**.

### ESEMPIO Sequenza di attivazione

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:



$$Tw = (P1, P2, P3) = 11 / 3 = 3.6$$

Il dispositivo che rende possibile la sospensione di un processo ancora in esecuzione allo scadere del **time slice**, è il **Real-time clock (RTC)**. Esso non è altro che un chip impiantato sulla scheda madre contenente un cristallo di quarzo che viene fatto oscillare in modo estremamente stabile con segnali elettrici: tali oscillazioni scandiscono il tempo generando periodicamente delle interruzioni da inviare al **sistema operativo**.

### AREA digitale

Periodo dell'RTC

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.

## Algoritmo MLFQ

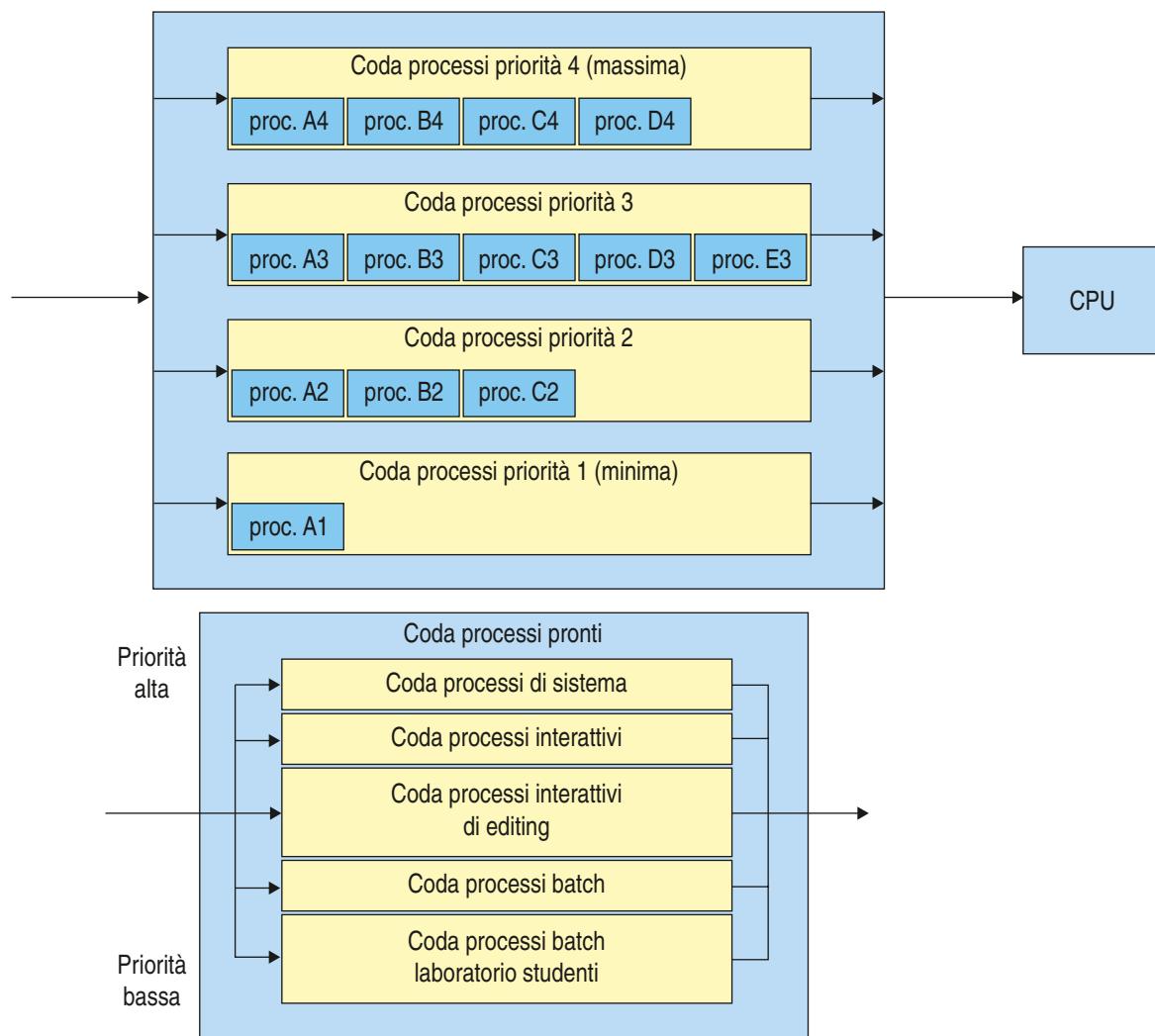
L'algoritmo **MLFQ**, acronimo di **M**ultiple **L**evel **F**eedback **Q**ueues, si ottiene dalla combinazione di un **FCFS** con un algoritmo a priorità.

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 7/8

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).

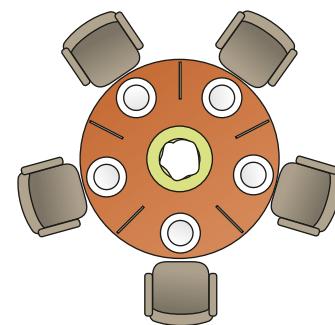
Per comprendere la necessità di **sincronizzazione** presente nei SO vediamo di seguito un famoso esempio proposto da **Dijkstra** nel 1965, conosciuto con il nome di “filosofi a cena”.

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 trattendo 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 **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



### 1. Risposta multipla

**1 Nei sistemi multitasking:**

- a. sono presenti più processori
- b. più programmi evolvono contemporaneamente
- c. più processi sono contemporaneamente in memoria
- d. più processi sono in esecuzione contemporaneamente

**2 Il PCB non contiene:**

- a. Identificatore unico (PID);
- b. Stato corrente;
- c. Program counter;
- d. Stack
- e. Registri;
- f. Priorità;
- g. Puntatori alla memoria del processo;

**3 Dallo stato di pronto un processo può passare:**

- a. allo stato di esecuzione
- b. allo stato di terminazione
- c. allo stato di attesa
- d. allo stato di inizio

**4 Dallo stato di attesa un processo può passare:**

- a. allo stato di esecuzione
- b. allo stato di terminazione
- c. allo stato di pronto
- d. allo stato di inizio

**5 Nei sistemi batch si cerca di:**

- a. massimizzare throughput
- b. minimizzare il tempo medio di risposta
- c. minimizzare il tempo di attesa
- d. minimizzare turnaround

**6 Nei sistemi interattivi si cerca di:**

- a. massimizzare throughput
- b. minimizzare il tempo medio di risposta
- c. minimizzare il tempo di attesa
- d. minimizzare turnaround

**7 La SRTF è:**

- a. la FCFS con pre-emptive
- b. la FCFS non pre-emptive
- c. la SJF con pre-emptive
- d. la SJF non pre-emptive

**8 La starvation dei processi si verifica:**

- a. con politiche FCFS
- b. con politiche SJF
- c. con politiche SRTF
- d. con politiche a priorità
- e. con la round robin
- f. sempre

### 2. Vero o falso

- 1 I SO cercano di minimizzare il throughput del sistema.
- 2 Lo scheduling dei job consiste nei meccanismi utilizzati per la scelta dei programmi disco.
- 3 Un programma è l'evoluzione di un processo.
- 4 Il multitasking si ottiene su architetture parallele di sistema.
- 5 Con PCB si indica il Program Control Block.
- 6 Il descrittore di un processo contiene anche il codice del programma.
- 7 Dallo stato di pronto un processo può passare allo stato di attesa.
- 8 Dallo stato di esecuzione un processo si sospende se gli manca una risorsa per evolvere.
- 9 Le operazioni eseguite per il cambio di contesto hanno generalmente la durata di 1 nanosecondo.
- 10 L'algoritmo SJF è un caso particolare dell'algoritmo FCFS.
- 11 La starvation è dovuta alla mancanza di sincronizzazione tra processi.
- 12 Il time-slice ha generalmente la durata di 10 nanosecondi.



AREA digitale



# La gestione della memoria

**In questa lezione 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.

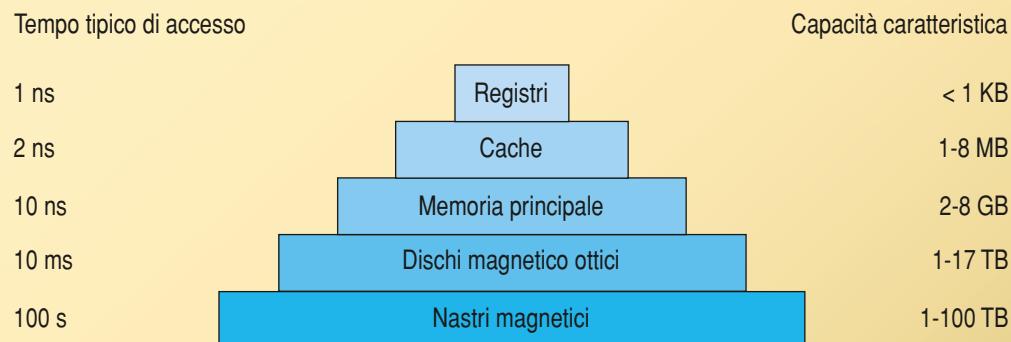
Ricordiamo una celebre frase di **Bill Gates**, fondatore della Microsoft, pronunciata nel 1985, che si rivelò profondamente errata: *"Chi mai avrà bisogno di più di 640 K di memoria centrale?"*. In sostanza si può affermare che la memoria centrale è una risorsa che "non basta mai"!

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 **deallocate**;
- 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.

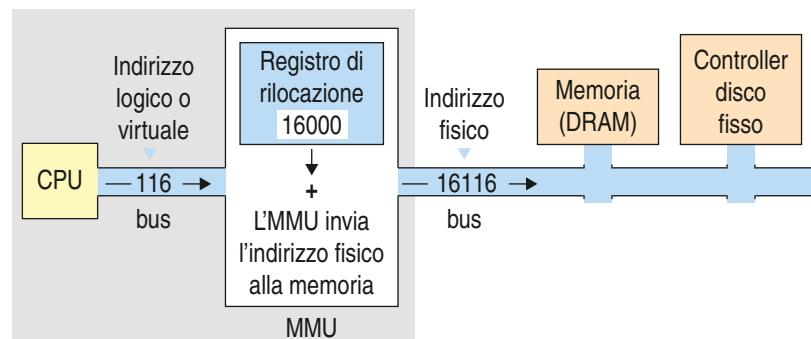
L'assegnazione degli indirizzi è quindi incompleta: vengono cioè generati degli **indirizzi relativi (indirizzo logico)** e all'atto del caricamento vero e proprio questi vengono trasformati in **indirizzi assoluti (indirizzo fisico)**.

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**.

Il dispositivo hardware che associa gli indirizzi virtuali agli indirizzi fisici è il **Memory Management Unit** (o **MMU**): si può osservare nello schema seguente come il valore contenuto nel *registro di rilocazione* viene sommato a ogni indirizzo generato dai processi utente nel momento stesso in cui l'indirizzo viene inviato alla memoria.



La formula generale è quindi:

$$\text{indirizzo fisico} = \text{indirizzo logico} + \text{offset}$$

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

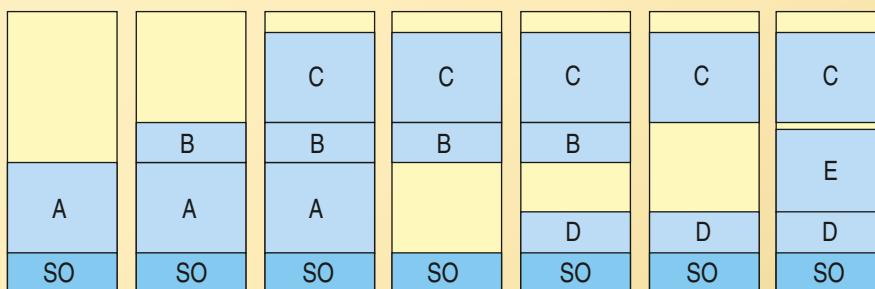
Il calcolo degli indirizzi logici viene effettuato nella fase di **linking** dei programmi; il passaggio dall'indirizzo logico a quello fisico avviene nella fase di **binding** che può essere fatta in momenti diversi:

- ▶ 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: il 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

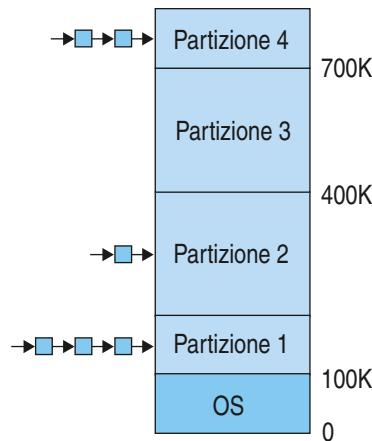
Nello schema a **partizione fissa** la dimensione della partizione viene definita all’atto dell’inizializzazione del sistema, quindi staticamente, e viene creata una tabella dove si memorizza lo stato delle partizioni, indicando quali sono libere e quali occupate (e da chi sono occupate).

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 interna



### Frammentazione esterna

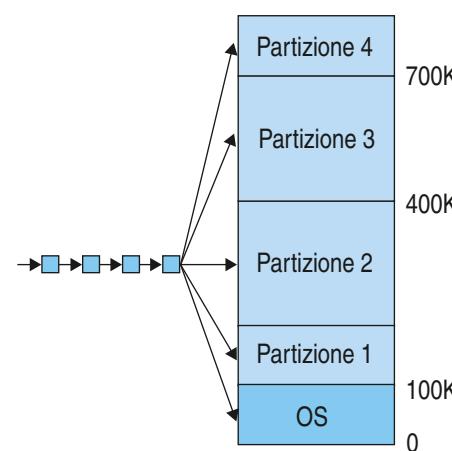
La dimensione di un processo può essere più grande di una qualunque partizione esistente anche se, nel complesso, la memoria associata a tutte le partizioni è sufficiente per contenerla.



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 suddividendo 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.

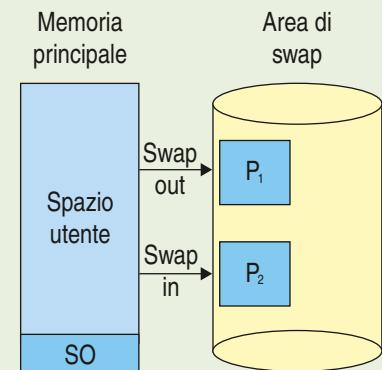
Le simulazioni hanno dimostrato che le strategie migliori sono le prime due, in particolar modo la **first-fit**, che è la più veloce.



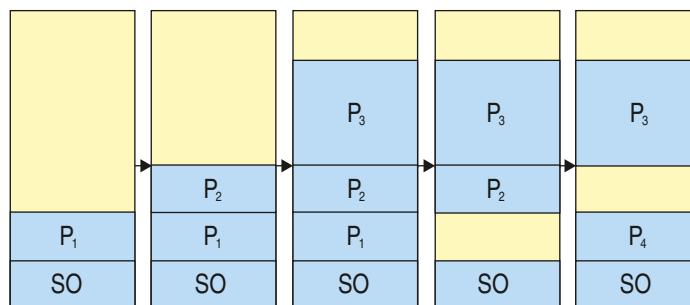
**Zoom su...**

### AREA DI SWAP

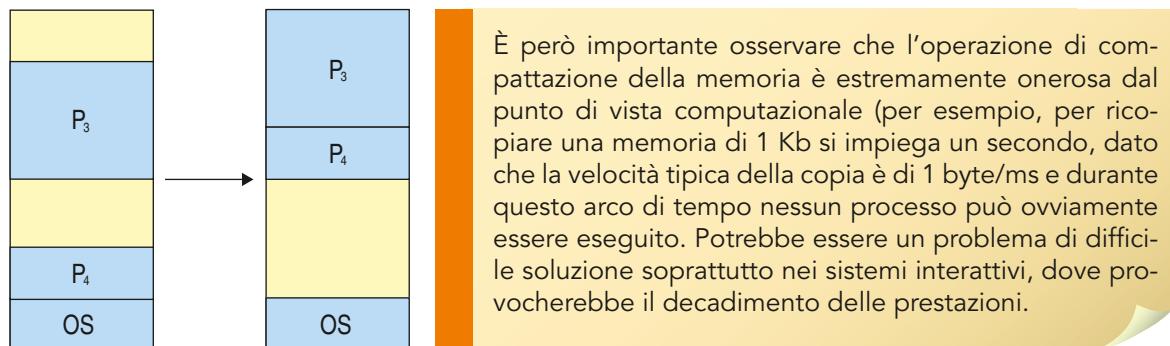
Per ottimizzare l'utilizzo della memoria generalmente il partizionamento dinamico fa uso del disco come area di appoggio (**backing store** o **swap**) per i processi che per un qualsiasi motivo non sono in grado di continuare e passano nello stato di wait: per esempio, un processo che effettua una chiamata 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.



## ■ 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**, **segmentazione** e **tecniche ibride** che uniscono i vantaggi di entrambe le soluzioni.

Il principale motivo dell'introduzione di queste tecniche è quello di ridurre la frammentazione della memoria, cioè della quantità di memoria che risulta non utilizzabile, ma l'introduzione di una tecnica di allocazione non contigua può portare a un altro fondamentale vantaggio: caricare in memoria solo le parti di programma che effettivamente sono utili per l'evoluzione del processo in quel dato istante e liberare le altre parti per poter caricare nuovi task.

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 a essa vicina: quindi è sufficiente avere caricato un "pezzo" di codice mentre il resto può rimanere sulla memoria di massa.

Mediante questa tecnica è quindi possibile recuperare altro spazio della memoria centrale per massimizzare il numero di processi in esecuzione contemporanea.

Inoltre 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

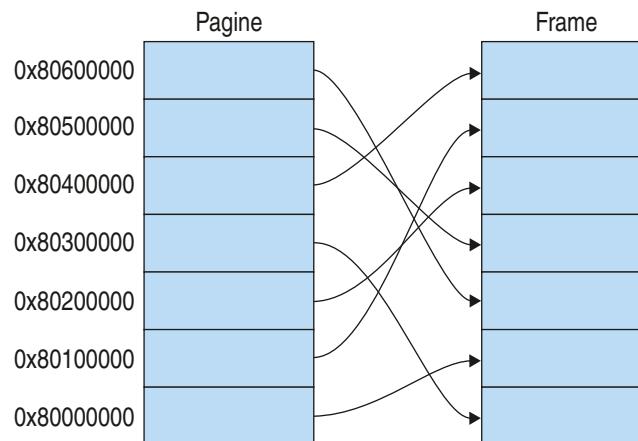
Il primo metodo che descriviamo per risolvere i problemi appena descritti è la **paginazione**.

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**).



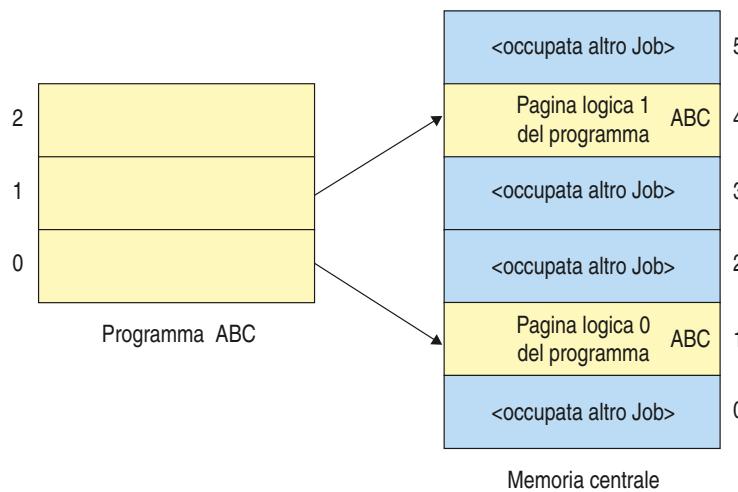
Naturalmente il numero di **pagine logiche** (la cui somma costituisce la dimensione del programma) può essere diverso dal numero di **pagine fisiche** (la cui somma è la dimensione della memoria):

- 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 **maggior**e del numero delle pagine fisiche (libere) il programma verrà caricato in memoria parzialmente.

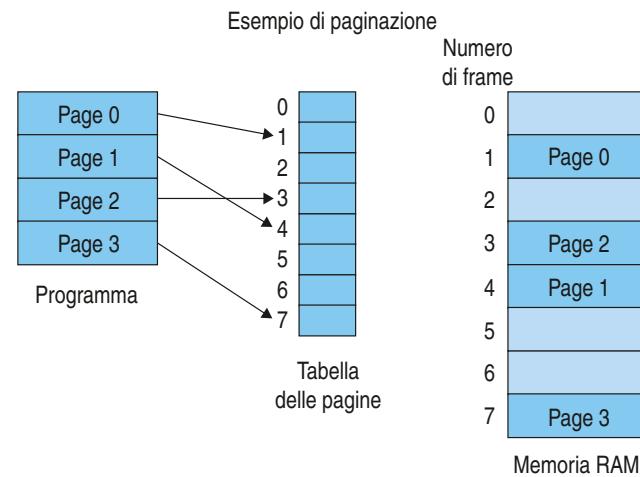
Non necessariamente le pagine fisiche devono essere contigue e quindi le pagine logiche possono trovarsi "mischiate" nella memoria centrale: naturalmente affinché un processo possa sfruttare questa tecnica di gestione condizione necessaria è che il codice sia **rilocabile dinamicamente**.

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**.

Nella figura seguente si vede come il nuovo processo non viene completamente caricato in RAM, ma solo due pagine logiche trovano spazio libero e tale spazio **non è contiguo**.



Il SO cura una **tabella delle pagine** dove il numero stesso di **página física** 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 página è occupata o libera, ed eventualmente l'ID del processo che la occupa.

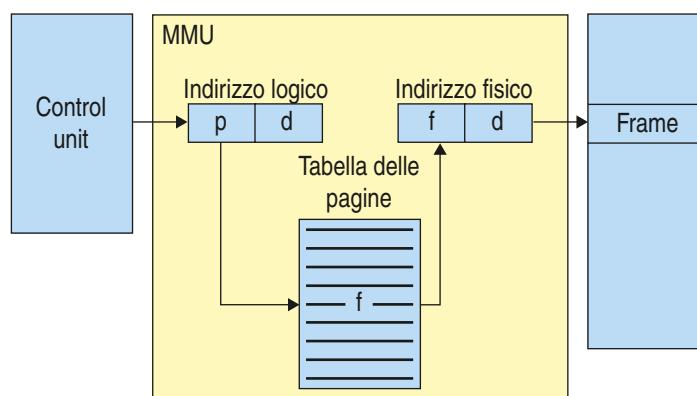


Inoltre, anche a ogni processo viene associata una **tabella delle pagine** specifica dove sono indicati quali segmenti di codice sono caricati in RAM e quali su disco.

La gestione delle pagine rientra tra i compiti del SO mentre la traduzione da indirizzo virtuale a indirizzo fisico viene effettuata da **MMU**, precedentemente già descritto, 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.

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 pagina** l'indirizzo fisico (**f**) al quale deve essere sommato l'offset (**d**).



**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.

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.

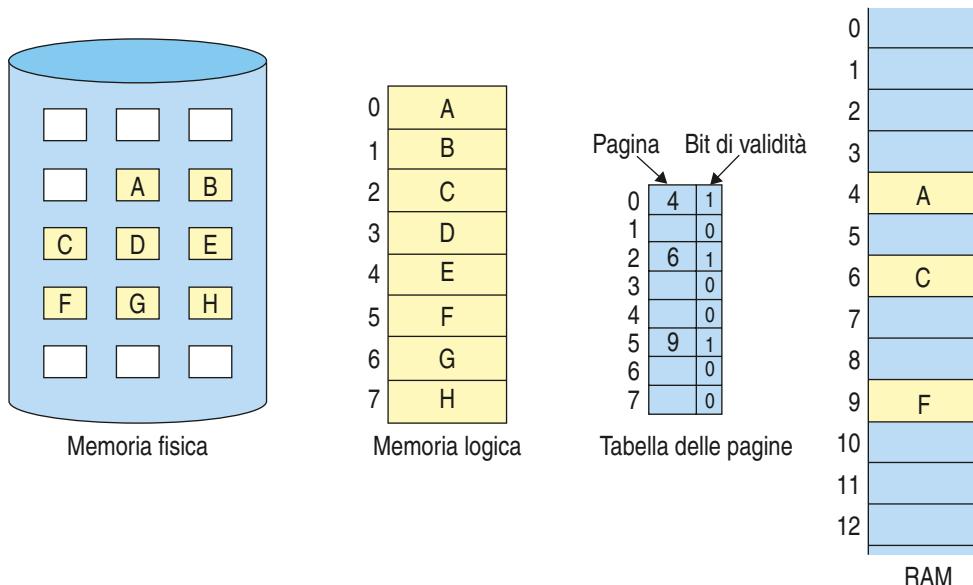
### Page fault

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'ecccezione 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.

A ogni pagina logica viene aggiunto nella tabelle delle pagine un bit, detto **bit di validità**, che a seconda dello stato della pagina cambia di valore (1 = pagina in memoria, 0 = pagina non in memoria).



Quindi quando una pagina viene caricata in memoria il suo bit di validità viene posto a 1 per indicare la sua presenza in memoria: nella fase di traduzione dall'indirizzo virtuale all'indirizzo fisico si consulta la **tabella delle pagine** e se il **bit di validità** è uguale a 0 si ha un **page fault**.

Naturalmente non sempre è presente in memoria un frame libero: in tale caso è necessario "fare spazio" e viene richiamato un **algoritmo di sostituzione** per selezionare un **frame vittima**: la pagina vittima viene scritta sul disco e la pagina richiesta viene letta nel frame appena liberato dopo aver effettuato l'aggiornamento corrispondente delle tabelle delle pagine e dei frame.

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.

## ■ 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.

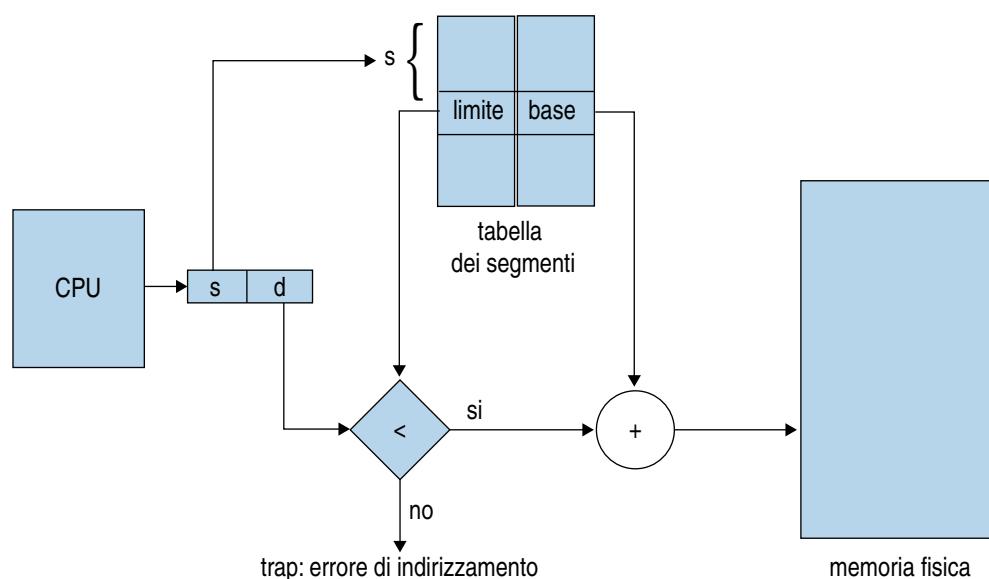


### FRAME E SEGMENTI

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.

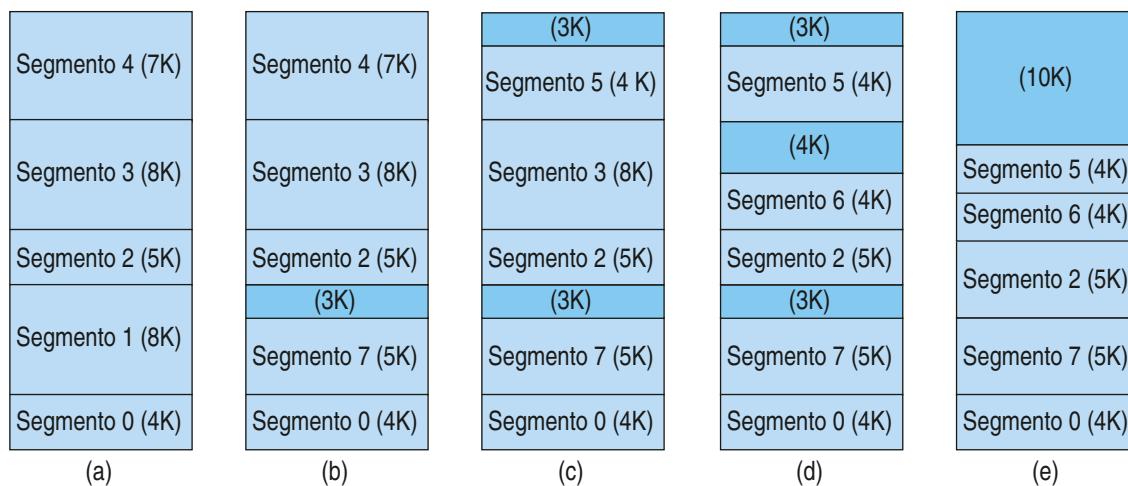
Con la **segmentazione** si ottengono i seguenti **vantaggi**:

- ▶ 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.

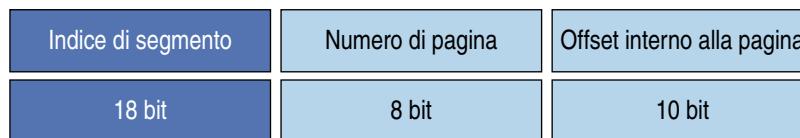


### FRAME E SEGMENTI PAGINATI

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.



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.



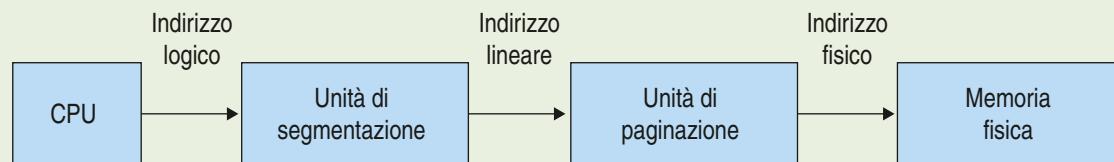
## Zoom su...

### 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

### 1. Risposta multipla

**1 Tra i compiti del memory manager abbiamo:**

- a. sapere sempre quali parti della memoria centrale sono in uso
- b. individuare il programma da caricare in memoria
- c. scegliere quale parte di memoria allocare ai processi
- d. gestire lo swapping tra la memoria principale e il disco

**2 L'indirizzo logico:**

- a. viene assegnato a partire dalla cella 0
- b. viene rilocato dal linker
- c. viene trasformato in indirizzo fisico dal linker
- d. viene trasformato in indirizzo assoluto dal loader

**3 L'acronimo MMU significa:**

- a. Memory Management Utility
- b. Memory Maker Unit
- c. Memory Management Unit
- d. Memory Maker Utility

**4 Nella rilocazione statica:**

- a. viene sommato a ogni istruzione l'offset
- b. viene utilizzato il registro base
- c. l'indirizzo base viene sommato durante l'esecuzione
- d. l'indirizzo base viene sommato durante la fase di link

**5 Il page fault si verifica quando:**

- a. le pagine di memoria sono tutte occupate
- b. in memoria non ci sono pagine libere
- c. il sistema operativo non trova la pagina da caricare
- d. un programma indirizza una pagina non presente in memoria

**6 Quale tra le seguenti non è una strategia che risolve il problema dell'allocazione dinamica della memoria centrale?**

- a. first-fit
- b. best-fit
- c. good-fit
- d. worst-fit

**7 Quale tra le seguenti affermazioni è falsa in merito alla paginazione?**

- a. gestisce ogni volta solo piccole porzioni di memoria
- b. la memoria fisica è divisa in blocchi chiamati frame o pagine fisiche
- c. il programma è diviso in blocchi di uguale dimensione detti pagine
- d. il numero di pagine logiche è uguale al numero delle pagine fisiche
- e. il codice deve essere rilocalizzabile dinamicamente
- f. si risolve il problema della frammentazione esterna

### 2. Associazione

**1 Assegna a ogni memoria le sue velocità caratteristiche.**

- a. Registri
- b. Nastri magnetici
- c. Memoria RAM
- d. Cache
- e. Dischi magneto-ottici

- 1. 10 msec
- 2. 1 nsec
- 3. 100 sec
- 4. 2 nsec
- 5. 10 nsec

### 3. Vero o falso

- 1 La rilocazione statica avviene all'atto del caricamento in memoria.
- 2 L'offset si ottiene sommando all'indirizzo di base i riferimenti.
- 3 Nella rilocazione dinamica l'indirizzo assoluto viene determinato dal loader.
- 4 Lo swapping avviene quando la memoria RAM libera principale non è sufficiente.
- 5 Il gestore della memoria ha il compito di deallocare la memoria quando serve.
- 6 Il loader preleva un processo dalla coda di entrata e lo colloca nella lista dei processi pronti.
- 7 Il compilatore generalmente genera degli indirizzi relativi (indirizzo fisico).
- 8 Nella rilocazione dinamica viene utilizzato il registro base per contenere l'offset.

- V F
- V F
- V F
- V F
- V F
- V F
- V F
- V F



# La memoria secondaria: il file system

In questa lezione impareremo...

- ▶ il concetto di file
- ▶ la struttura di una directory
- ▶ il modello client-server

## ■ Introduzione

Un'altra risorsa fondamentale di un sistema di calcolo è la **memoria secondaria**: è il supporto di memorizzazione non volatile di grosse dimensioni necessario a un sistema di calcolo per effettuare la memorizzazione permanente dei programmi e dei dati.

Questa tipologia di memoria coinvolge movimenti meccanici e quindi il tempo di accesso è particolarmente elevato: l'accesso a essa è quindi un punto critico e l'utilizzo non efficiente di questa risorsa porta a criticità sui tempi del sistema.

Nel corso degli anni la **memoria secondaria** ha avuto una evoluzione sia in termini tecnologici ma anche dal punto di vista della sua organizzazione: i primi calcolatori, infatti, utilizzavano i **nastri magnetici** come unità di memorizzazione aventi come caratteristica propria l'**accesso sequenziale** che richiede di “scorrere” tutto il nastro per trovare il dato desiderato.

Il passaggio ai **dischi magnetici (hard disk)** ha comportato l'introduzione di nuove tecniche di organizzazione dei dati e nuovi problemi, anche perché per memorizzare un dato su di un HD necessariamente si deve fare riferimento a posizioni fisiche del disco rigido.

Per evitare di gestire singolarmente i blocchi fisici i dati vengono organizzati in **file** e la loro gestione è demandata a un componente specifico del **SO**: il **file system**.



## FILE SYSTEM

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 di tipo magnetico o magnetoottico, in grado di memorizzare grandi volumi di dati.

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 **SO** che maggiormente comunica con l’utente e spesso viene identificato con il **SO**: si colloca come interfaccia tra l’utente e i dispositivi dando una rappresentazione astratta e unificata dei dispositivi fisici effettivi presenti nel sistema.

## AREA digitale



Il primo calcolatore con disco magnetico

## ■ 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 associati 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;



◀ **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**. ▶

- ▶ **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.

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 azzerà 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.

### AREA digitale



Condivisione di file

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.

## 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**.

### AREA digitale



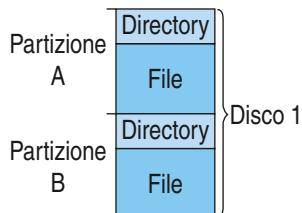
Accesso indicizzato

## ■ 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 **Albero delle directory**

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**.

Cartelle	Nome	Dimensione	Tipo	Data ultima modifica
Disco locale (C:)				
\$AVG	CATEGORIE	3 KB	File SQL	20/04/1995 13.10
29fdf32edc7f111b5d49	DEMO	1,329 KB	File DB5	26/01/1998 3.02
CA_APPSW	ORDINI	11 KB	File SQL	20/04/1995 13.08
CAVO	PRODOTTI	17 KB	File SQL	20/04/1995 13.16
CAVO20				
BIN				
CAIDT				
DEMO				
VENDITE				

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**.

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).



## 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
Esegibile	exe, com, bin	Programma pronto per l'esecuzione
Codice oggetto	obj, o	Codice oggetto non ancora fuso in un esegibile
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**.



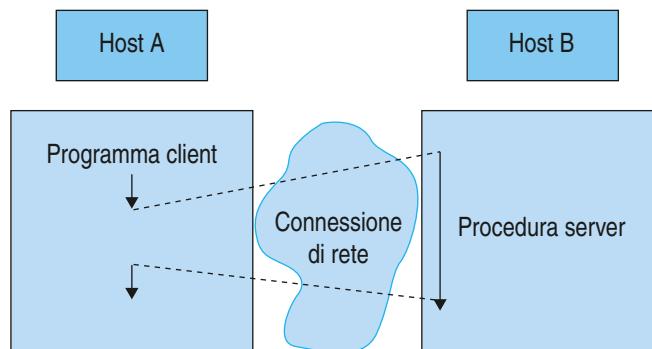
### HOST, SERVER E CLIENT

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 eseguito su un computer remoto
È invocato dall'utente	Inizia l'esecuzione con l'inizializzazione del sistema
Richiede la comunicazione con il server	Rimane in attesa di richieste dai client
Spedisce una richiesta	Spedisce una risposta (dati o file)
Può accedere a più servizi, ma uno alla volta	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, roture 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.

### AREA digitale



Flag dei diritti nei file Unix

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



### 1. Risposta multipla

**1** Nel descrittore del file non è presente:

- a. il nome del file
- b. la dimensione
- c. il tipo di file
- d. la directory dove è memorizzato
- e. i flag di protezione

**2** In una directory sono presenti:

- a. solo file dello stesso tipo
- b. solo file con diversa dimensione
- c. solo file dello stesso utente
- d. solo file con diverso nome

**3** Nella copia di un file il suo nome deve essere

- a. cambiato sempre
- b. cambiato solo su un altro dispositivo
- c. cambiato solo nella stessa directory
- d. cambiato solo in altre directory

**4** Quale tra le seguenti operazioni è primitiva, cioè non utilizza altre operazioni:

- a. posizionamento
- b. cancellazione
- c. scrittura
- d. lettura
- e. ricerca

**5** Un file si dice condiviso quando:

- a. contiene dati di più utenti
- b. è stato scritto contemporaneamente dai più utenti
- c. può essere utilizzato contemporaneamente da più processi
- d. viene utilizzato da più calcolatori

**6** Nel partizionamento dei dischi si ha:

- a. un MBR per ogni volume
- b. un MBR per ogni settore
- c. un MBR nel primo settore
- d. un MBR per ogni partizione

**7** In un sistema client-server

- a. in ogni rete esiste un solo server
- b. in ogni rete esiste un solo client
- c. in ogni rete esiste una coppia client-server
- d. ogni macchia può essere client o server

**8** Con diritti su di un file si intende:

- a. la paternità del suo creatore
- b. i permessi per compiere le operazioni
- c. la rivendicazione economica su di un file
- d. nessuna delle precedenti affermazioni



### 2. Vero o 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 memorizzati 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

# ESERCITAZIONI DI LABORATORIO 1

## LA SHELL DEI COMANDI DI WINDOWS

### ■ La shell di sistema

Il sistema operativo mette a disposizione degli utenti un ambiente chiamato **shell** di sistema, grazie al quale possiamo interagire con la macchina.



◀ La **shell** consente agli utenti di interagire con il sistema operativo in modalità a linea di comando, i comandi vengono cioè impartiti al **prompt di sistema**. È un componente principale di un sistema operativo e il suo nome deriva dalla lingua inglese dove shell significa "guscio". Gli utenti possono impartire i comandi grazie alla shell che rappresenta quindi l'involucro del **kernel** di sistema: grazie a questa caratteristica viene anche chiamata interfaccia utente. ►

Esistono svariati tipi di shell, che possono essere suddivisi in due categorie principali: **testuali** e **grafiche**. Le shell testuali vengono solitamente indicate semplicemente come **shell** o come **comandi terminale**: l'utente interagisce attraverso un'interfaccia a riga di comando (**CUI**). Nel caso delle shell grafiche è comune fare riferimento ai cosiddetti **desktop environment**, che forniscono agli utilizzatori un ambiente grafico da cui è possibile gestire file e avviare programmi.

In sintesi le shell di Windows sono principalmente tre: **command.com**, **cmd.exe** e **powershell.exe**, mentre le shell testuali di Linux sono principalmente: **Bash** (Bourne-Again Shell), **Korn shell** e **C shell**.



## Zoom su...

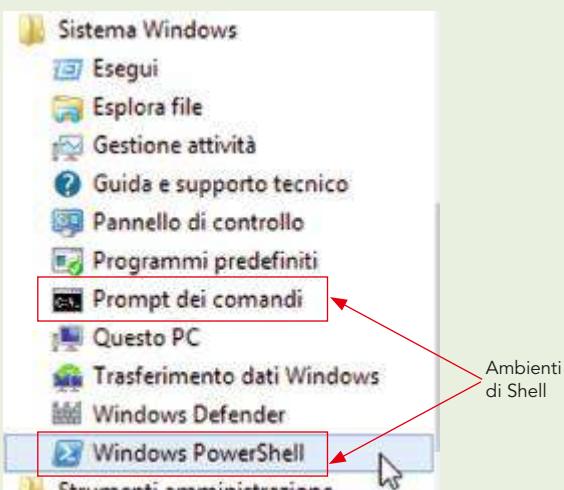
### POWERSHELL DI WINDOWS

Il **prompt dei comandi** di Windows rappresenta una delle più note shell a riga di comando. La shell di Windows è normalmente rappresentata dal file **cmd.exe** per le versioni **NTFS**, mentre per le versioni più obsolete (**FAT** e **FAT32**) è il file **command.com**. Esiste anche una particolare shell per Windows chiamata **PowerShell** che rispetto alle precedenti offre un ambiente di **scripting** avanzato ed è orientata agli oggetti, i comandi vengono chiamati **cmdlet** (commandlet) e restituiscono oggetti al posto di flussi di testo.

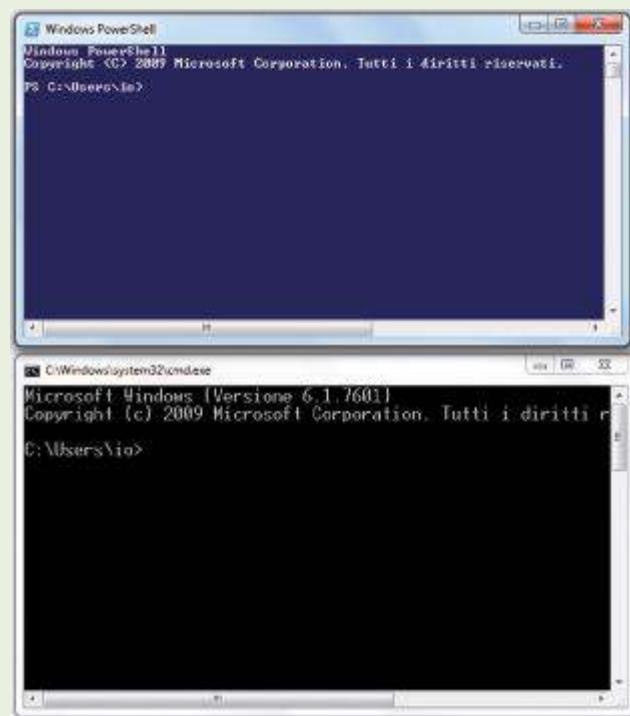
Per entrare negli ambienti di shell esiste il menu di Windows chiamato **Sistema Windows** in cui appaiono le due icone **Prompt dei comandi** e **Windows PowerShell**:

Per attivare gli ambienti di shell l'utente può, in alternativa rispetto al menu grafico, digitare il comando **cmd.exe** oppure **powershell.exe** dal menu **esegui** ottenibile con la combinazione di tasti **Finestra + R**.

A partire dalla versione **7** di Windows la **PowerShell** è sempre presente nel sistema: nelle versioni precedenti di Windows era necessario invece installare a parte il pacchetto PowerShell.



La seguente figura mostra la finestra di **PowerShell** rispetto alla finestra di **Prompt dei Comandi** di **Windows**:



Quando il sistema mostra il **Prompt dei comandi** significa che è in attesa di un comando da parte dell'utente. Come possiamo notare la PowerShell aggiunge **PS** davanti al messaggio di sistema.



## Prova adesso!

- Entrare in ambiente shell di Windows
- Applicare un comando nell'ambiente di shell

- 1** Entra in ambiente **Prompt dei comandi** di Windows e prova i seguenti comandi (ricorda che al termine di ciascun comando devi premere il tasto Enter (Invio) per mandarlo in esecuzione:

DIR  
CLS  
VOL  
TIME  
DATE

- 2** Entra in ambiente **PowerShell** di Windows e prova i seguenti comandi (ricorda che al termine di ciascun comando devi premere il tasto Enter (Invio) per mandarlo in esecuzione:

ls  
cls  
volume  
Get-Time  
Get-Date

- 3** Annota le differenze che hai riscontrato.

## ■ I comandi di shell

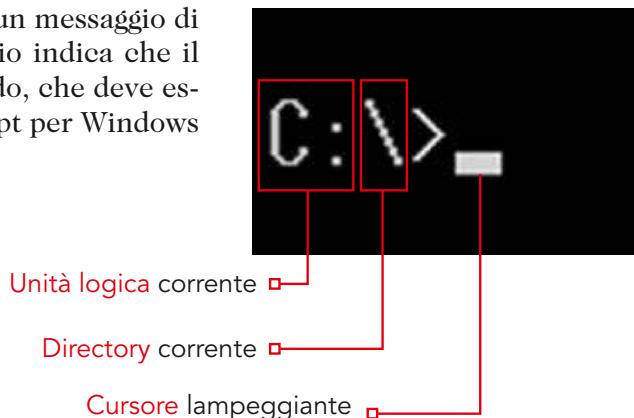
L'ambiente **prompt dei comandi** mostra un messaggio di sistema chiamato **prompt**. Tale messaggio indica che il sistema è pronto per ricevere un comando, che deve essere digitato dall'utente. Il classico prompt per Windows è illustrato a fianco.

Possiamo individuare i seguenti elementi:

- **C:** è il nome dell'unità di memoria di massa, in questo caso il disco fisso. L'unità evidenziata viene chiamata **unità corrente** o **unità di default**: è l'unità sulla quale agiranno i nostri comandi (se non indichiamo diversamente);
- **\** indica la **root** (radice), che in questo caso è la **directory corrente**;
- **>** il simbolo di maggiore serve unicamente come separatore tra il **prompt** e il comando vero e proprio che verrà digitato dall'utente in corrispondenza del **cursore lampeggIANte**.

I comandi di shell di Windows, chiamati anche **comandi DOS**, possono essere suddivisi in due categorie:

- comandi **interni**
- comandi **esterni**

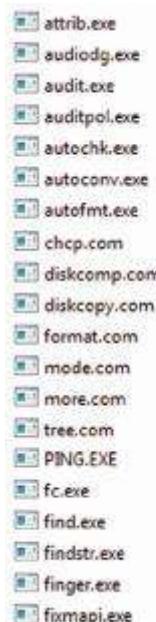


I primi vengono chiamati così in quanto caricati in memoria durante il bootstrap; non necessitano quindi di un file specifico per essere utilizzati ed eseguiti. Per i comandi esterni invece esiste un file eseguibile (.exe o .com) specifico per ciascun comando. Normalmente risiedono nella directory \windows\system32, come mostrato nella figura a fianco che ne mostra solo alcuni: ►

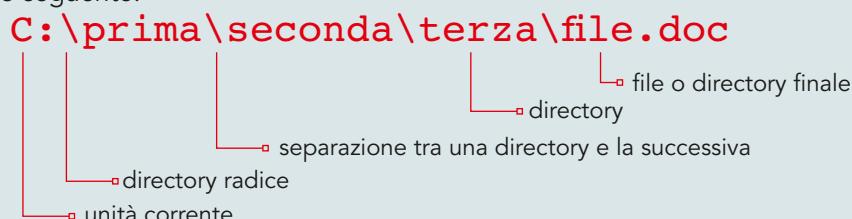
Alcuni esempi di questi comandi sono rappresentati dai file **find.exe** per il comando **find** oppure **format.com** per il comando **format**, che verranno illustrati più avanti. La sintassi di un comando esterno non dipende dal sistema operativo ma dal programmatore che ha prodotto il file eseguibile. In generale la sintassi generica di un comando esterno è la seguente:

```
[drive:\path\] nome_comando parametri
```

in cui **drive:\path\** rappresenta il ◀ **percorso** ▶ che contiene il file eseguibile.



◀ Con il termine **percorso** o **pathname** si indica la sequenza di directory necessarie per raggiungere un file. I percorsi di file e directory si separano **back slash** (\), come riportato nell'esempio seguente:

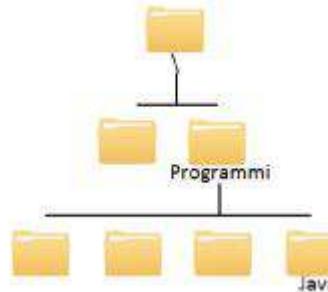


### ESEMPIO **Il percorso di un file**

Ad esempio il file **elenco.txt** possiede il percorso **C:\Programmi\Java\elenco.txt**, come possiamo notare dalla figura: ►

File	Modifica	Visualizza	Strumenti	?
Organizza	Includi nella raccolta	Condividi con	Masterizza	Nuova cartella
OS (C:)		Name		Ultima modifica
Programmi		jdk1.7.0_07		29/11/2012 17:40
Java		jdk1.7.0_09		29/11/2012 17:50
		jre7		29/11/2012 17:48
		elenco.txt		27/09/2012 16:39

Il percorso del file potrebbe essere così schematizzato: ►



I percorsi dei file o delle directory possono essere suddivisi in:

- **assoluti**;
- **relativi**.

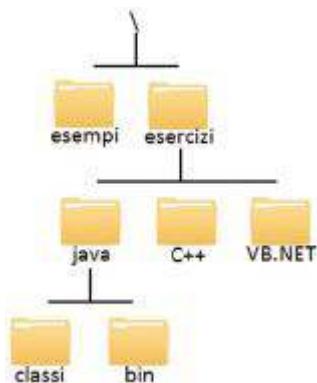
I **percorsi assoluti** identificano un file o una directory partendo sempre dall'unità logica o dalla **root**. Ad esempio **C:\dos\prova.txt** è un percorso assoluto: indica che il file **prova.txt** è contenuto nella directory **dos** che è figlia della root dell'unità logica del disco fisso.



◀ La **root** indica la directory radice dell'albero, si indica con il simbolo **\** (back slash) e viene creata automaticamente alla formattazione del disco. La struttura delle directory e i file costituiscono quello che viene chiamato **file system**. ►

### ESEMPIO **Percorso assoluto**

Consideriamo il seguente percorso di cartelle all'interno del disco fisso:



Prima di tutto ricordiamoci che le directory possiedono una struttura gerarchica ad **albero**, in cui ogni livello è in parentela con un altro. Ad esempio le directory **esempi** ed **esercizi** sono figlie della **root** e sorelle tra di loro, mentre le directory **java**, **C++** e **VB.NET** sono figlie della directory **esercizi**.

Un file chiamato **prova.txt**, contenuto nella directory **C++**, possiede il seguente percorso assoluto:

```
\esercizi\C++\prova.txt
```

Un file chiamato **eser.com** è presente nella directory **bin**, quindi possiede il seguente percorso assoluto:

```
\esercizi\java\bin\eser.com
```

Possiamo leggere così il primo percorso:

*“Il file prova.txt è contenuto nella directory C++ che è figlia della directory esercizi che a sua volta è figlia della root”.*

Possiamo leggere così il secondo percorso:

*“Il file eser.com è contenuto nella directory bin che è figlia della directory java che è figlia della directory esercizi che a sua volta è figlia della root”.*

Possiamo anche affermare che:

*“La directory bin è nipote di esercizi” oppure che “La directory java è nipote della root”.*

I **percorsi relativi** identificano un file o una directory partendo dal punto in cui ci si trova all'interno della memoria di massa, non iniziano quindi mai con una **unità logica** oppure con il simbolo della **root** (\).



◀ Un **percorso** è sempre **relativo** quando si riferisce alla directory corrente. Il corrispondente percorso assoluto viene calcolato dal sistema. ►

Possiamo affermare che un percorso relativo è sempre legato alla posizione che occupiamo all'interno dell'albero di directory di un supporto di memoria, sia esso un disco fisso, una pen drive oppure una qualunque unità logica.

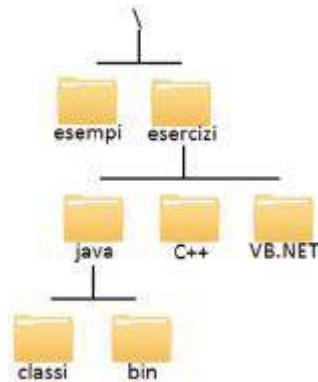
Un percorso relativo inizia sempre con un nome di file o di directory oppure con un **carattere speciale** che può essere:

- (punto)
- .. (punto punto)

Il **punto**, chiamato **current**, rappresenta la directory in cui ci troviamo mentre il **punto punto**, chiamato **parent**, indica il percorso di livello superiore.

### ESEMPIO **Percorso relativo**

Consideriamo il seguente percorso di cartelle all'interno del disco fisso:



Un file chiamato **prova.txt**, contenuto nella directory **java**, possiede il seguente percorso relativo se siamo posizionati nella directory **bin**:

**..\prova.txt**

Se siamo posizionati nella directory **C++** il percorso relativo di **prova.txt** è:

**..\java\prova.txt**

Infine siamo posizionati nella directory **esempi**, il percorso relativo di **prova.txt** è:

**..\esercizi\java\prova.txt**

Possiamo leggere così il primo percorso:

*“Sali di un livello e quindi trovi il file prova.txt”.*

Possiamo leggere così il secondo percorso:

*“Sali di un livello, quindi entra nella directory java e quindi trovi il file prova.txt”.*

Possiamo leggere così il terzo percorso:

*“Sali di un livello, quindi entra nella directory esercizi, quindi entra nella directory java e qui trovi il file prova.txt”.*

Come abbiamo notato per riferirci a una directory “sorella” di quella in cui siamo posizionati è necessario prima di tutto salire di un livello mediante il carattere speciale **parent** (..), quindi indicare il nome della directory figlia separato dallo slash.

## ■ Il comando PATH

Prima di tutto partiamo dal presupposto che un comando esterno può essere eseguito conoscendone il percorso, oppure posizionandoci nella directory in cui esso è presente. Ad esempio se ci posizioniamo nella directory **c:\windows\system32** e digitiamo il comando **CALC** verrà eseguito il file **calc.exe** in quanto è presente in tale percorso.



Ma cosa accade se ci spostiamo in un altro percorso (per fare questo basta digitare il comando **cd** seguito dal percorso da raggiungere)? Il file non viene eseguito e appare un **messaggio di errore**:

```
C:\Windows\System32>cd \
C:\>calc
"calc" non è riconosciuto come comando interno o esterno,
un programma eseguibile o un file batch.
```

Questo messaggio di errore viene mostrato in quanto nella directory in cui ci troviamo, in questo caso la **root**, non è presente il file **calc.exe** necessario per l'esecuzione del comando.

Scrivendo il nome del file senza anteporre alcun percorso, abbiamo usato un percorso relativo, che indica la directory in cui ci troviamo.

Tuttavia esiste una **variabile d'ambiente** che “informa” il sistema operativo della presenza di altre directory in cui potrebbe essere presente il comando esterno.



◀ Le **variabili d'ambiente** contengono alcune impostazioni relative al funzionamento del sistema operativo. Si tratta di stringhe di caratteri che contengono informazioni quali percorsi di file, unità disco, o nomi di file. Ad esempio la variabile d'ambiente **TEMP** contiene il nome della directory all'interno della quale le applicazioni installate potranno salvare eventuali file temporanei. ▶

Tale variabile d'ambiente si chiama **path** e contiene un elenco di percorsi all'interno dei quali il sistema andrà alla ricerca dei file necessari per eseguire i comandi digitati quando questi non dovessero essere presenti nella directory corrente. I valori di alcune variabili d'ambiente, tra cui anche path, vengono impostati in fase di avvio del sistema operativo.

Solitamente tali variabili sono inizializzate da parte del sistema operativo con valori di default che possono tuttavia venire modificati dall'utente in qualunque momento.

Il comando **PATH** seguito dall'Invio mostra a video l'elenco dei percorsi contenuti nella variabile d'ambiente **path**:

```
C:\Users\io>PATH  
PATH=c:\;c:\Programmi;c:\Java;c:\Java\files;c:\Program Files\Java
```

Come possiamo notare tra i percorsi non vi è **C:\windows\system32**, pertanto il sistema non è conoscenza della presenza di tale percorso. Per modificare i percorsi della variabile **path** dobbiamo digitare il comando seguito da tutti i percorsi che vogliamo indicare, separati tra di loro dal punto e virgola. Aggiungiamo quindi il percorso **C:\windows\system32**:

```
C:\Users\io>PATH c:\Windows\System32  
C:\Users\io>_
```

A questo punto potremo far eseguire il comando **calc.exe** posizionandoci in qualsiasi altro percorso, ad esempio nella **root**:



### Prova adesso!

- Utilizzare il comando **path**
- Riconoscere i percorsi relativi e assoluti
- Eseguire un comando esterno

**NB:** Affinché questo esercizio possa essere eseguito è necessario aver installato il pacchetto Office.

- 1 Verifica che dalla root del prompt dei comandi non sia possibile eseguire il file **winword.exe**.
- 2 Aggiungi quindi alla variabile d'ambiente **path** il seguente percorso:  
**C:\Program Files (x86)\Microsoft Office\Office14**.
- 3 Verifica ora che sia possibile eseguire il file.

# ESERCITAZIONI DI LABORATORIO 2

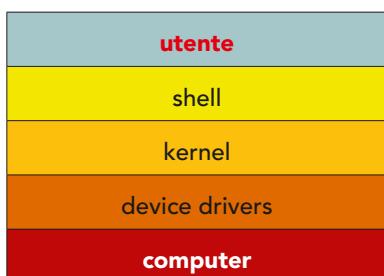
## I COMANDI PRINCIPALI

### ■ Il file system

Il sistema operativo è formato da un insieme di programmi e di procedure che si occupano di caricare in memoria i programmi, di gestire il dialogo con le periferiche e di permettere l'interfacciamento con l'utente. I principali componenti sono:

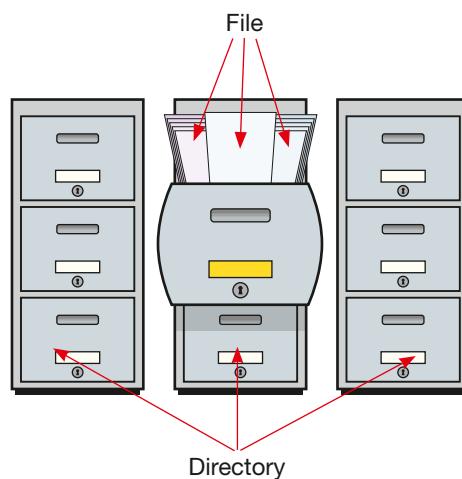
- 1 il **kernel** o nucleo;
- 2 i **device drivers**;
- 3 la **shell**.

Il **kernel**, dall'inglese “nocciole”, indica il nucleo del sistema che si occupa di gestire i programmi, garantendone il caricamento in memoria e l'esecuzione. I **device drivers** consentono la gestione di alcuni elementi hardware fondamentali, come ad esempio il disco oppure le periferiche di input e output. Il **kernel** opera con i **device driver** per poter utilizzare il disco e accedere a esso. La **shell** infine rappresenta il programma che consente il dialogo con l'utente in modalità interattiva. Possiamo rappresentare **shell**, **kernel** e **device drivers** come tre strati di astrazione fra utente e computer.



Il **file system** è quella parte del sistema operativo che gestisce i file e le directory. La **struttura ad albero** delle directory è definita dal file system, così come le regole dei **nomi** dei file.

I **nomi** e le **estensioni** dei file e delle directory possono contenere lettere, numeri e alcuni simboli: underscore (\_), meno (-), parentesi graffe({}), punto esclamativo (!) e tilde (~). Non possono tuttavia contenere quasi nessun altro simbolo, in particolare non possono contenere spazi e punteggiatura. La struttura ad albero, come visto nella scorsa lezione, prevede che la directory principale sia la **root** (radice) e il simbolo a essa associato sia il back slash. Il carattere di **back slash** viene tuttavia utilizzato anche per separare i nomi delle directory successive all'interno di un percorso.



## Zoom su...

### L'HELP DEI COMANDI

Per ottenere un piccolo **help** di aiuto di ogni singolo comando è sufficiente scrivere il comando seguito dallo **slash** e quindi dal **punto di domanda**. Ad esempio per sapere quali parametri possiede il comando dir digitiamo **dir /?**, la figura seguente ne mostra solo una piccola parte:

```
C:\>dir/?
Mostra l'elenco dei file e delle sottodirectory in una directory.

DIR [unità:][[percorso][nomefile]] [/A[[lattributi]] [/B] [/C] [/D] [/L] [/N]
 [/O[[ordinamento]] [/P] [/Q] [/R] [/S] [/T[[campo dati]] [/W] [/X] [/4]

[unità:][[percorso][nomefile]]
    Specifica unità, directory e/o file da elencare.

/A      Visualizza i file con gli attributi specificati.
attributi   D Directory          R File sola lettura
            H File nascosti       A File archivio
            S File di sistema     I File non indicizzati
            L Reparse point       - Prefisso per negare l'attributo
            Utilizza lista file senza intestazione o informazioni di
            riepilogo.
/C      Visualizza il separatore delle migliaia nelle dimensioni dei
        file. Impostazione predefinita. Utilizzare /-C per disabilitarla.
/D      Come /N ma con i file ordinati per colonna.
```

Il comando **help** invece mostra l'elenco di tutti i comandi utilizzabili all'interno del prompt dei comandi, la figura seguente ne mostra solo una piccola parte:

```
C:\>help
Per ulteriori informazioni su uno specifico comando, digitare HELP nome_comando
ASSOC Visualizza o modifica le associazioni alle estensioni dei file.
ATTRIB Visualizza o modifica gli attributi del file.
BREAK Attiva o disattiva il controllo esteso di CTRL+C.
BCDEDIT Imposta le proprietà nel database di avvio per il controllo del
caricamento avvio.
CACLS Visualizza o modifica gli elenchi di controllo di accesso
(ACL) dei file.
CALL Richiama un programma batch da un altro.
CD Visualizza il nome della directory corrente o consente
di passare a un'altra directory.
CHCP Visualizza o imposta il numero di tabella codici attiva.
```

## ■ Il comando prompt

Il comando **prompt** modifica l'aspetto del **prompt dei comandi**, cioè del messaggio di sistema della shell. Possiamo in tal modo personalizzare l'aspetto del prompt per mostrare ad esempio il nome della directory corrente, l'ora e la data, la versione del sistema operativo. La sintassi è la seguente:

```
prompt [testo]
```

Il testo specifica qualsiasi testo o informazione che si desidera includere nel **prompt di sistema**. Nel seguente elenco sono riportate le combinazioni di caratteri che è possibile specificare nel parametro testo al posto o in aggiunta di una qualsiasi stringa di caratteri con una breve descrizione delle informazioni che ogni combinazione aggiunge al prompt dei comandi:

Testo	Significato
\$Q	mostra il simbolo di uguale (=)
\$\$	mostra il simbolo del dollaro (\$)
\$T	mostra l'ora corrente
\$D	mostra la data corrente
\$P	mostra l'unità e il percorso corrente
\$V	mostra la versione del sistema operativo
\$N	mostra la sola unità logica corrente
\$G	mostra il simbolo di maggiore (>)
\$L	mostra il simbolo di minore (<)
\$_	aggiunge un invio a capo
\$E	mostra il simbolo di escape (freccina)



### Prova adesso!

• Utilizzare il comando prompt

- 1 Modifica il prompt in modo che scriva il tuo nome seguito dal simbolo di escape.
- 2 Modifica il prompt in modo che mostri la data e l'ora su due righe separate.
- 3 Modifica il prompt in modo che mostri l'unità logica e la versione di sistema operativo.
- 4 Riporta il prompt alla normalità, per fare questo utilizza il comando seguente:

```
prompt $P$G
```

## ■ Le unità logiche

Le **unità logiche** sono specificate con lettere seguite dal simbolo dei due punti (ad esempio **C:**). Le lettere **A:** e **B:** sono associate a supporti ormai in disuso (floppy disk), mentre la prima lettera utile è la lettera **C:**. Quando si hanno più dischi o più partizioni a disposizione, per identificare univocamente un file è necessario indicare anche in quale disco, o per meglio dire, unità logica, è collocato. Ad esempio il file **prova.doc** si trova nella directory **esempi** dell'unità logica **E:**, questo significa che il suo percorso assoluto è il seguente:

```
E:\esempi\prova.doc
```

Per cambiare il disco corrente, cioè per passare da una unità logica a un'altra, dobbiamo digitare la lettera dell'**unità logica** da raggiungere seguita dai due punti (:) e quindi **Invio**. Ad esempio per passare all'unità logica di una pen drive (E:) dobbiamo digitare: ►

```
C:\>e:  
E:\>
```

I principali comandi per i supporti di memoria (dischi e periferiche di memoria) sono i seguenti:

- **fdisk** (per la creazione della **partizione**);
- **format** (per la **formattazione**);
- **subst** (per la creazione di **unità logiche virtuali**);
- **chkdisk** (per il recupero di unità danneggiate).

La sintassi di **fdisk** è assai semplice, basta scrivere il nome e appare un menu che consente di **creare** o **eliminare** una partizione. Non permette tuttavia di modificare le partizioni esistenti.

Il comando **fdisk** manipola le partizioni e pertanto va usato con estrema attenzione: un uso improprio di questo comando può dare luogo a conclusioni catastrofiche per il contenuto del disco rigido. Nelle versioni più recenti di Windows è infatti stato tolto dai comandi esterni e deve essere installato appositamente.

Il comando **format** consente di formattare una partizione, possiede numerosi parametri che, tra l'altro, consentono di effettuare una formattazione **rapida** oppure **completa**. La sintassi è:

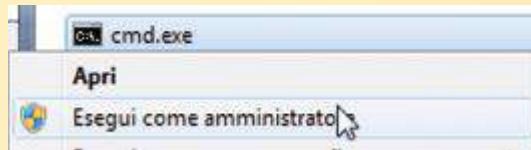
```
format [unità logica]
```

Vediamo un esempio.

#### ESEMPIO **Formattazione del disco rigido**

È meglio fare una premessa: questa operazione è **distruttiva**, quindi eseguitela solo se effettivamente sicuri e consapevoli di ciò che state facendo. Ricordate che i dati presenti andranno perduti a meno che non si disponga di speciali utilities di recupero dati.

Per poter utilizzare il comando **format** devi eseguire la shell di Windows come **amministratore**, facendo click sul programma **cmd.exe** con il tasto destro e scegliendo **Esegui come amministratore**:



Questa è una opzione di Windows necessaria per proteggere i nostri dati.

Prima di tutto per **formattare** un **HDD** (**Hard Disk Drive**) dobbiamo necessariamente collegarlo alla scheda madre del PC (cavo **SATA** oppure **IDE** per i modelli meno recenti). Esistono diverse procedure di formattazione, noi andremo adesso a formattare il disco in modo **Lento** e **Veloce** per verificarne le differenze.

Prima di eseguire tale processo è consigliato sempre eseguire un backup di tutti i vostri più importanti dati, pena la perdita degli stessi.

Il prerequisito fondamentale è quello di verificare il nome esatto dell'unità logica che andiamo a formattare: per fare questo andiamo su **Computer** di Windows e verifichiamo la lettera associata all'hard disk da formattare. In questo caso abbiamo verificato che il disco possiede unità logica **E:**. La sintassi per effettuare una **formattazione lenta**, che elimina tutti i dati in modo irrecuperabile, è la seguente:

```
format E: /FS:NTFS
```

La sintassi per effettuare una **formattazione veloce**, che elimina solo alcuni dati, è la seguente:

```
format E: /FS:NTFS /Q
```

## AREA digitale



Correggere lo stato dei supporti di memoria

## AREA digitale



Creare un'Unità virtuale

## ■ I comandi di gestione delle directory

I comandi della shell di Windows non fanno distinzione tra caratteri maiuscoli e minuscoli, quindi non sono **case sensitive**. I principali comandi per la gestione delle directory sono i seguenti:

- ▶ **cd** o **chdir**
- ▶ **md** o **mkdir**
- ▶ **rd** o **rmdir**

Il primo comando serve per **cambiare** la directory di lavoro o directory corrente, in quanto significa **change directory** (**cd**). La sintassi è:

```
cd [percorso directory]
```

Ovviamente la directory in cui vogliamo posizionarci deve esistere, in caso contrario il sistema segnalera l'errore seguente:

```
C:\Users\io>cd pluto
Impossibile trovare il percorso specificato.
```

Per cui volendo entrare nella root da qualunque directory siamo posizionati digitiamo:

```
cd \
```

Il percorso può essere espresso sia in modo **assoluto**, come in questo caso, che **relativo**. Per posizionarci nella directory di **livello superiore**, se disponibile, attraverso percorso relativo, digitiamo:

```
cd ..
```

Il secondo comando serve per **creare** una nuova directory, in quanto significa **make directory** (**md**). La sintassi è:

```
md [percorso directory]
```

Per cui volendo creare una nuova directory di nome **prova**, all'interno della cartella **system32** di **windows**, digitiamo il percorso completo:

```
md \windows\system32\prova
```

La **root** non deve essere mai creata, così come non può essere mai rimossa. Viene automaticamente creata ogni volta che si formatta un supporto.

Il percorso può essere espresso sia in modo **assoluto**, come in questo caso, che **relativo**. Per creare una directory di nome **esempio**, come figlia di quella corrente, digitiamo:

```
md esempio
```

Il terzo comando serve per **eliminare** una directory, in quanto significa **remove directory** (**rd**). La sintassi è:

```
rd [percorso directory]
```

Per cui volendo eliminare la directory di nome **prova**, all'interno della cartella **system32** di **windows**, digitiamo il percorso completo:

```
rd \windows\system32\prova
```

Il percorso può essere espresso sia in modo **assoluto**, come in questo caso, che **relativo**. Per eliminare una directory di nome **esempio**, come sorella di quella corrente, digitiamo:

```
rd ..\esempio
```

Mediante questo comando non possiamo tuttavia eliminare le directory che possiedono file al loro interno oppure che contengono altre directory a loro volta. Tuttavia questa limitazione può essere superata aggiungendo il parametro **/S** che fa in modo che vengano eliminati tutti i file o le sottodirectory presenti in essa.

## AREA digitale



Esempi di comandi CD, MD e RD

- Utilizzare i comandi per le directory



### Prova adesso!

- Crea l'albero seguente di directory: ►
- Adesso entra nelle seguenti directory attraverso percorsi assoluti: **root, mate, doc, esercizi**.
- Posizionati ora entra nelle seguenti directory attraverso percorsi relativi: **root, scuola, mate, scuola, informatica, scuola, elettronica, relazioni, doc, materiale**.
- Elimina la directory **relazioni**.
- Elimina la directory **scuola** con un solo comando.



A questi comandi si aggiunge il comando **dir** che mostra il contenuto della directory corrente o del percorso indicato. Esso mostra l'elenco di file e directory analogamente alla **visualizzazione dettagli** di Windows, in quanto mostra per ciascun file **data** e **ora** di creazione e la **dimensione**. Le directory sono individuabili in quanto precedute dalla dicitura **<DIR>**. La sintassi è:

```
dir [percorso directory]
```

Ad esempio vediamo il contenuto della root del disco fisso:

```
C:\>dir
Il volume nell'unità C è OS
Numero di serie del volume: 7437-FA8F

Directory di C:\

06/09/2012 18:19 <DIR>          AsusVibeData
18/12/2013 19:55 <DIR>          Dev-Cpp
18/12/2013 21:17 <DIR>          Dev-Pas
06/09/2012 05:55          15.061 devlist.txt
18/10/2012 16:20 <DIR>          java
31/05/2011 05:26          19 K73SD_K73E_K73SV_WIN7.40
06/09/2012 05:35 <DIR>          NvidiaLogs
27/02/2014 18:43 <DIR>          office2003
14/07/2009 05:20 <DIR>          PerfLogs
08/10/2014 21:47 <DIR>          Program Files
16/10/2014 16:01 <DIR>          Program Files (x86)
14/12/2012 16:58 <DIR>          Word2007
01/07/2014 12:13 <DIR>          www
04/06/2013 22:09 <DIR>          xampp
                           2 File      15.080 byte
                           12 Directory 402.130.640.896 byte disponibili
```

Per mostrare l'elenco a pagine possiamo utilizzare i parametri **/w** e **/p**. Il primo mostra l'elenco su più colonne e il secondo attende la pressione di un tasto per la visualizzazione della pagina successiva.



## Zoom su...

### IL COMANDO TREE

Attraverso il comando **tree** possiamo ottenere l'interpretazione grafica dell'**albero di directory**, a partire dal percorso indicato come parametro. La sintassi è assai semplice:

```
tree [percorso] [/F]
```

Con il parametro **/F** vengono mostrati anche tutti i file contenuti in ciascuna directory. Se il percorso viene omesso viene adottata la directory corrente.

Proviamo, ad esempio, a mostrare l'albero di directory della directory **\dev-cpp**:

```
tree \dev-cpp
```

Il risultato è il seguente: ▶

Come possiamo notare le nuove sottodirectory vengono mostrate procedendo verso destra, analogamente a quanto accade usando l'ambiente GUI di Windows.

```
C:\>tree \dev-cpp
Elenco del percorso delle cartelle per il volume OS
Numero di serie del volume: 7437-FA8F
C:\DEV-CPP
    +-- bin
        +-- Examples
            +-- FileEditor
            +-- Hello
            +-- Jackpot
            +-- MDIAppl
            +-- OpenGL
            +-- Simpwin
            +-- WinAnim
            +-- WinMenu
            +-- WinTest
        +-- Help
        +-- Icons
        +-- include
            +-- C++
                +-- 3.4.2
                    +-- backward
                    +-- bits
                    +-- debug
                    +-- ext
                    +-- mingw32
                        +-- bits
            +-- ddk
            +-- GL
            +-- sys
        +-- Lang
        +-- lib
            +-- debug
            +-- gcc
                +-- mingw32
                    +-- 3.4.2
                        +-- include
```

## ■ Gli attributi dei file

Vediamo i comandi più utili per la gestione dei file. Prima di tutto vediamo come ottenere gli **► attributi ►** dei file, attraverso il comando **attrib**, la cui sintassi è la seguente:

```
attrib [+R|-R] [+A|-A] [+S|-S] [+H|-H] [+I|-I] [percorso file]
```

Gli attributi dei file indicano alcune caratteristiche quali ad esempio se il file è nascosto oppure se è a sola lettura. Gli attributi sono i seguenti:



◀ Ogni file system prevede degli **attributi** per i file, quelli di NTFS sono: **sola lettura, di sistema, nascosto, archivio e indicizzato.** ▶

- ▶ **R** Attributo di file di **sola lettura** (se attivo il file non può essere modificato)
- ▶ **A** Attributo di file di **archivio** (in Windows è ininfluente, è rimasto solo per compatibilità con il DOS)
- ▶ **S** Attributo di file di **sistema** (se attivo il file può essere caricato in memoria all'avvio)
- ▶ **H** Attributo di file **nascosto** (se attivo impedisce di visualizzare i file, per visualizzare tali file usare dir/a)
- ▶ **I** Attributo di file **non indicizzato** (se attivo non colloca il file nell'elenco dei file indicizzati)

I parametri del comando sono i seguenti:

- ▶ **+R** aggiunge l'attributo di sola lettura (**Read Only**)
- ▶ **-R** toglie l'attributo di sola lettura (**Read Only**)
- ▶ **+A** aggiunge l'attributo archivio
- ▶ **-A** toglie l'attributo archivio
- ▶ **+S** aggiunge l'attributo di sistema (**System**)
- ▶ **-S** toglie l'attributo di sistema (**System**)
- ▶ **+H** aggiunge l'attributo nascosto (**Hidden**)
- ▶ **-H** toglie l'attributo nascosto (**Hidden**)
- ▶ **+I** aggiunge l'attributo di file non indicizzato (**Index**)
- ▶ **-I** toglie l'attributo di file non indicizzato (**Index**)

Possono essere impostati anche più attributi in un unico comando. Ad esempio per rendere un file sia nascosto che a sola lettura impostiamo il seguente comando:

```
attrib +H +R esempio.doc
```

Possiamo verificare che il file è stato impostato mediante il comando **attrib** sul file:

```
attrib esempio.doc
```

### ESEMPIO Assegnare un attributo a un file

In questo caso vogliamo assegnare l'attributo di file **nascosto** e **non indicizzato** a un file di nome **prova.txt**. Il comando è il seguente:

```
attrib +h +i prova.txt
```

Verifichiamo di aver assegnato gli attributi al file con il comando seguente:

```
attrib prova.txt
```

Inoltre possiamo anche verificare che il file non sia più visualizzabile mediante il comando **dir**. Vediamo l'esecuzione di quanto indicato:

```
C:\Users\io>attrib +h +i prova.txt
C:\Users\io>attrib prova.txt
A H I C:\Users\io\prova.txt
```

Il comando **attrib \*.\*** mostra gli attributi di tutti i file contenuti nella directory corrente, anche di quelli nascosti.

Adesso togliamo l'attributo nascosto al file **prova.txt**:

```
C:\Users\io>attrib -h prova.txt
C:\Users\io>attrib prova.txt
A I C:\Users\io\prova.txt
```



## Prova adesso!

• Modificare gli attributi di un file

- 1 Assegna a un file a tua scelta gli attributi di sistema, sola lettura e nascosto.
- 2 Verifica di aver assegnato tali attributi.
- 3 Togli gli attributi impostati nel punto 1.

## ■ I comandi per i file

Vediamo i principali comandi che permettono di gestire i file:

- **ren** (cambia il nome a un file);
- **del** (cancella uno o più file);
- **type** (mostra il contenuto di un file in formato ASCII).

### ESEMPIO *Gestiamo i file*

Il comando **type** visualizza il contenuto di un file, tuttavia il risultato sarà illeggibile se non si tratta di un file di testo in **formato ASCII**. Vediamo come visualizzare il contenuto del file **elenco.txt** mediante il comando:

```
type elenco.txt
```

```
C:\>type elenco.txt  
Mario  
Luca  
Andrea  
Marina  
Carla  
Sandro  
  
C:\>
```

Proviamo adesso a cambiare nome al file, per fare questo usiamo il comando **ren**, seguito dai nomi del file: prima si digita il **vecchio nome** (**elenco.txt**), quindi il **nuovo nome** (**amici.txt**) che dovrà assumere:

```
ren elenco.txt amici.txt
```

```
C:\>ren elenco.txt amici.txt  
C:\>_
```

Infine cancelliamo il file **amici.txt** con il comando **del**:

```
del amici.txt
```

```
C:\>del amici.txt  
C:\>_
```

Come possiamo notare non viene richiesta alcuna conferma di cancellazione.

## ■ I comandi di copia

I comandi che permettono di copiare i file sono sostanzialmente due:

- ▶ **copy**
- ▶ **xcopy**

Il primo è un comando **interno**, mentre il secondo è un comando **esterno**. Quest'ultimo, tuttavia, è del tutto identico al primo se non per alcune caratteristiche che lo rendono in grado di copiare anche interi alberi di directory con relative sottodirectory grazie al parametro **/S** posto dopo il nome. La sintassi dei due comandi è la seguente:

```
copy [origine] [destinazione] [/Y]  
xcopy [origine] [destinazione] [/S]
```

Dove l'**origine** indica il file o il gruppo di file da copiare mentre la **destinazione** è il percorso all'interno del quale verranno copiati i file indicati come primo parametro. Il parametro **/Y** chiede conferma di sovrascrittura.

I file da copiare possono essere più di uno se utilizziamo i **caratteri jolly** che verranno illustrati più avanti.

Nella **destinazione** possiamo anche includere un nome di file, in questo caso il file copiato cambierà nome. Se nella destinazione esiste già un file con lo stesso nome, esso verrà sovrascritto dal nuovo file, a meno che non venga utilizzato il parametro **/Y**.

Per copiare più file in un unico file destinazione, dobbiamo indicare tutti i file come parametri **origine** separati tra di loro dal simbolo più (+), e dobbiamo necessariamente specificare un nome per il file generato dalla combinazione. Il comando seguente unisce i tre file (**gennaio.doc**, **febbraio.doc** e **marzo.doc**) in un unico file chiamato **relaz.doc**:

```
copy gennaio.doc+febbraio.doc+marzo.doc relaz.doc
```

Se non viene specificato un file di destinazione, il comando unisce i file in un file unico con lo stesso nome del primo file specificato.

### ESEMPIO **Copia di file da una cartella a un'altra**

Siamo nella situazione seguente: ►

Vogliamo copiare il file **prova.txt** dalla directory **scuola** alla directory **casa**:

```
copy \scuola\prova.txt \casa
```

Se fossimo ad esempio posizionati già nella directory **scuola** potremmo usare il percorso relativo:

```
copy prova.txt ..\casa
```

Se poi volessimo cambiare nome al file trasformandolo in **tesina.txt** dovremmo digitare il seguente comando:

```
copy \scuola\prova.txt \casa\tesina.txt
```

Adesso proveremo a copiare il file **esempio.txt** dalla directory **esercizi** in **materiale**, avendo **informatica** come **directory corrente**:

```
copy esercizi\esempio.txt ..\materiale
```

Il seguente comando copia tutto l'albero che inizia dalla directory **informatica** all'interno di **relazioni**:

```
xcopy \scuola\informatica \scuola\elettronica\relazioni /S
```



## ■ I comandi di rete

Vediamo alcuni comandi che consentono di operare con il dispositivo di rete.

### **IPCONFIG**

Il comando **ipconfig** permette di conoscere il proprio **indirizzo IP** della scheda ethernet. La sintassi è la seguente:

`ipconfig [/all]`

Il parametro `/all` mostra alcune informazioni aggiuntive, tra cui la descrizione del **DNS**, l'indirizzo **MAC**, l'abilitazione del **DHCP** ecc.

Il comando consente di ottenere importanti informazioni ed effettuare alcune configurazioni sulla propria rete. Mostra tutte le informazioni inerenti la scheda di rete: **indirizzo IP**, **maschera IP (subnet mask)**, indirizzo del **gateway**, cioè del router che consente l'accesso a Internet. Eseguendolo si ottiene:

```
C:\Users\io>ipconfig
Configurazione IP di Windows

Scheda Ethernet Connessione di rete Bluetooth:
  Stato supporto: . . . . . : Supporto disconnesso
  Suffisso DNS specifico per connessione:

Scheda Ethernet Connessione alla rete locale (LAN):
  Stato supporto: . . . . . : Supporto disconnesso
  Suffisso DNS specifico per connessione:

Scheda LAN wireless Connessione rete wireless:
  Suffisso DNS specifico per connessione:
  Indirizzo IPv4 . . . . . : 192.168.0.4 → Indirizzo IP della scheda
  Subnet mask . . . . . : 255.255.255.0 → Maschera IP della scheda
  Gateway predefinito . . . . . : 192.168.0.1 → Indirizzo IP del gateway
```

Il comando **ipconfig** possiede molti altri parametri che interessano soprattutto i sistemisti e gli esperti di rete, tuttavia vediamoli nel dettaglio:

<code>/all</code>	Visualizza le informazioni complete sulla configurazione
<code>/release</code>	Rilascia l' <b>indirizzo IP</b> per la scheda specificata
<code>/renew</code>	Rinnova l' <b>indirizzo IP</b> per la scheda specificata
<code>/flushdns</code>	Svuota la cache del resolver <b>DNS</b>
<code>/registerdns</code>	Aggiorna tutti i <b>lease DHCP</b> e registra di nuovo i nomi <b>DNS</b>
<code>/displaydns</code>	Visualizza il contenuto della cache del resolver <b>DNS</b>
<code>/showclassid</code>	Visualizza tutti gli <b>ID classe DHCP</b> consentiti per la scheda
<code>/setclassid</code>	Modifica l' <b>ID classe DHCP</b>



**Zoom su...**

### ALTRI COMANDI DI RETE

Esistono numerosi altri comandi di rete richiamabili all'interno della shell di Windows, vediamo l'elenco dei principali:

<code>arp</code>	Mostra la tabella arp
<code>route print</code>	Visualizza la tabella di routing dell'host
<code>tracert</code>	Segue il percorso di un pacchetto, hop per hop
<code>netstat</code>	Visualizza le connessioni TCP attive
<code>nslookup [dominio]</code>	Visualizza l'indirizzo IP associato al nome di dominio indicato interro-gando direttamente il server DNS

## PING

Il comando **ping** (*Packet Internet Grouper*) viene utilizzato per verificare se due schede di rete sono effettivamente connesse. Viene pertanto usato come strumento diagnostico per verificare se ad esempio due **host** sono collegati.



◀ Viene chiamato **host** ciascun computer appartenente a una rete che ospita risorse e servizi disponibili ad altri sistemi. ▶

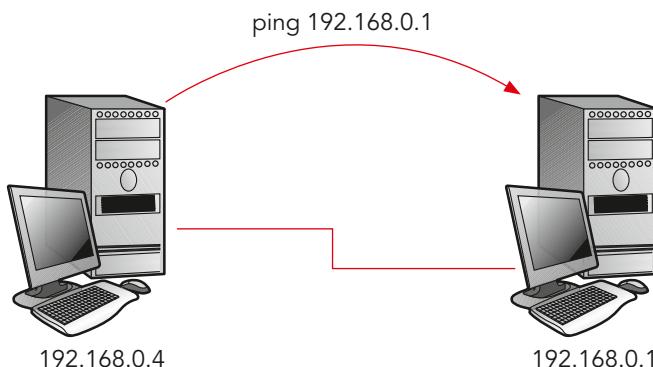
La sintassi è la seguente:

```
ping [indirizzo host destinazione]
```

Per essere precisi il comando misura il tempo, in millisecondi, impiegato da uno o più pacchetti a raggiungere un dispositivo di rete e a ritornare indietro all'origine.

### ESEMPIO Verifichiamo la connessione tra due host con ping

I questo esempio, due host possiedono indirizzo IP rispettivamente **192.168.0.4** e **192.168.0.1**.



Dall'host di indirizzo **192.168.0.4** vogliamo verificare l'effettiva connessione con l'host di indirizzo **192.168.0.1**, pertanto usiamo il comando seguente:

```
C:\>ping 192.168.0.1

Esecuzione di Ping 192.168.0.1 con 32 byte di dati:
Risposta da 192.168.0.1: byte=32 durata=105ms TTL=254
Risposta da 192.168.0.1: byte=32 durata=46ms TTL=254
Risposta da 192.168.0.1: byte=32 durata=1ms TTL=254
Risposta da 192.168.0.1: byte=32 durata=1ms TTL=254

Statistiche Ping per 192.168.0.1:
Pacchetti: Trasmessi = 4, Ricevuti = 4,
Persi = 0 (0% persi).
Tempo approssimativo percorsi andata/ritorno in millisecondi:
Minimo = 1ms, Massimo = 105ms, Medio = 38ms
```

Come possiamo notare il comando ha dato esito positivo, pertanto i quattro pacchetti inviati sono stati ricevuti e ritrasmessi. È possibile anche effettuare il **ping** con un indirizzo di Internet indicando il nome di dominio, ad esempio:

```
C:\>ping google.it

Esecuzione di Ping google.it [173.194.116.23] con 32 byte di dati:
Risposta da 173.194.116.23: byte=32 durata=35ms TTL=51
Risposta da 173.194.116.23: byte=32 durata=34ms TTL=51
Risposta da 173.194.116.23: byte=32 durata=36ms TTL=51
Risposta da 173.194.116.23: byte=32 durata=35ms TTL=51

Statistiche Ping per 173.194.116.23:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 34ms, Massimo = 36ms, Medio = 35ms
```

Come possiamo notare viene verificata la connessione con il dominio di google.it, che scopriamo avere **indirizzo IP 172.194.116.23**.



### Prova adesso!

- Utilizzare i comandi ipconfig e ping

- 1 Verifica l'indirizzo IP del tuo computer, con il comando corretto, e annotalo.
- 2 Prova adesso a verificare se, all'interno della tua aula computer, il tuo host è connesso al computer di un compagno di classe.

## AREA digitale



La gestione delle utenze

# ESERCITAZIONI DI LABORATORIO 3

## I CARATTERI JOLLY, REINDIRIZZAMENTO E PIPELINING

### ■ I caratteri jolly

I caratteri jolly, chiamati anche **metacaratteri**, vengono usati per **raggruppare** i file principalmente durante le operazioni di copia, visualizzazione e cancellazione. Vediamoli nel dettaglio:

- asterisco (\*);
- punto di domanda (?);
- punto (.).

Sostituiscono delle parti di nome o di estensione comuni a un gruppo di file, l'asterisco sostituisce una intera sequenza di caratteri di lunghezza qualsiasi, anche nulla, a partire dalla posizione in cui è inserito. Il carattere punto di domanda sostituisce invece un solo carattere in corrispondenza della posizione in cui è stato inserito. Il punto infine indica tutti i file.

#### ESEMPIO *Caratteri jolly nei comandi copy, del e dir*

Supponiamo di avere all'interno della stessa directory i seguenti file:

programmi.txt  
conti.bat  
paghe.dat  
prezzi.dat  
cassa.dat  
saldi.pdf

Vogliamo copiare tutti i file che iniziano con la lettera “p” e con qualunque estensione nella root del disco E:. Per fare questo sostituiamo il nome del file da copiare con il carattere jolly asterisco, quindi p\*.\*:

```
copy p*.* \
```

Ora invece vogliamo copiare tutti i file con estensione .bat e .dat nella directory \esempi. Possiamo raggruppare i file interessati in quanto hanno in comune gli ultimi due caratteri

dell'estensione, quindi `*.at`.

```
copy *.at \esempi
```

Adesso vogliamo copiare tutti i file che non hanno estensione (`*`) nella **root**.

```
copy *. \
```

Vogliamo copiare i file `paghe.dat`, `saldi.pdf` e `cassa.dat` nella directory `\esempi`. Come possiamo notare hanno in comune il secondo carattere del nome (`?a*.*`).

```
copy ?a* \esempi
```

Per copiare tutti i file della **root** nella directory corrente utilizziamo il carattere **punto** oppure, in alternativa, il simbolo `*.*`. La destinazione, in questo caso, può essere omessa in quanto si tratta della stessa directory in cui siamo posizionati. Tuttavia potremmo utilizzare, in alternativa, il carattere punto:

```
copy \*.*
```

oppure

```
copy \.
```

Per copiare tutti i file con estensione `.txt`, dalla directory corrente alla directory di livello **superiore**, utilizziamo il carattere jolly asterisco nel modo seguente: `*.txt`.

```
copy *.txt ..
```

Supponiamo di voler eliminare tutti i file `.com` dalla cartella `\esempi`. Per fare questo utilizziamo il comando `del` al quale applicare il carattere jolly `*.com`.

```
del \esempi\*.com
```

Supponiamo di voler visualizzare tutti i file di estensione `.exe` presenti nella directory `\windows\system32`. Per fare questo applichiamo i caratteri jolly al comandi `dir` (`*.exe`).

```
C:\Windows\System32>dir *.exe
Il volume nell'unità C è OS
Numero di serie del volume: 7437-F88F

Directory di C:\Windows\System32

25/10/2014 21:28      45.056 acount.exe
14/07/2009 03:38      40.448 AdapterTroubleshooter.exe
20/11/2010 15:24      122.880 aitagent.exe
14/07/2009 03:38      79.360 alg.exe
14/07/2009 03:38      17.920 appidcertstorecheck.exe
14/07/2009 03:38      146.944 appidpolicyconverter.exe
14/07/2009 03:38      24.064 ARP.EXE
14/07/2009 03:38      28.672 at.exe
14/07/2009 03:38      35.328 AtBroker.exe
14/07/2009 03:38      18.432 attrib.exe
20/11/2010 15:24      126.464 audiogd.exe
14/07/2009 03:38      64.000 auditpol.exe
20/11/2010 15:24      777.728 autochk.exe
20/11/2010 15:24      793.088 autoconv.exe
20/11/2010 15:24      763.904 autofmt.exe
14/07/2009 03:38      58.880 AxInstUI.exe
20/11/2010 15:24      175.616 bcdboot.exe
```

Supponiamo di voler visualizzare i file lunghi al massimo quattro caratteri con estensione `.exe`. Per fare questo utilizziamo quattro punti di domanda, seguiti dal punto e dall'estensione che in questo caso è nota.

```
C:\Windows\System32>dir ?????.exe
Il volume nell'unità C è OS
Numero di serie del volume: 7437-F88F

Directory di C:\Windows\System32

14/07/2009 03:38    79.360 alg.exe
14/07/2009 03:38    24.064 ARP.EXE
14/07/2009 03:38    28.572 at.exe
14/07/2009 03:38    918.528 calc.exe
14/07/2009 03:38    32.256 clip.exe
20/11/2010 15:24    345.088 cmd.exe
14/07/2009 03:39    24.064 comp.exe
14/07/2009 03:39    881.664 dccw.exe
14/07/2009 03:39    274.944 Dism.exe
14/07/2009 03:39    120.320 dwm.exe
14/07/2009 03:39    24.064 fc.exe
```



## Prova adesso!

- Applicare i caratteri jolly ai comandi dir e copy

- 1 Visualizza i file della directory \windows\system32 che hanno estensione .exe, il cui terzo carattere è una "t".
- 2 Copia nella root i file con estensione .dll lunghi al massimo cinque caratteri presenti nella directory windows\system32.

## ■ Il reindirizzamento

Prima di effettuare il **reindirizzamento** dei comandi dobbiamo conoscere come vengono individuati i diversi dispositivi, che possono essere così schematizzati:

Dispositivo	Descrizione
A: B:	Unità a disco floppy
C:	Prima unità a disco rigido
D: ... Z:	Altre unità o partizioni logiche utilizzabili per dischi rigidi, pen drive, CDROM ecc...
CON	Console cioè tastiera oppure schermo
PRN	Stampante in linea
LPT1, LPT2	Porte parallele
COM1, COM2	Porte seriali
NUL	Nessun dispositivo

Tutti i comandi visti finora operano su di un **input** che può essere indicato in modo **esplicito**, come nel caso dell'origine e della destinazione del comando **copy**, oppure **implicito**, come nel caso dell'**output** del comando **dir**. Più in generale il dispositivo da cui il comando riceve l'input viene chiamato **standard input**, mentre il dispositivo a cui il comando invia i dati per l'output viene chiamato **standard output**.

Ad esempio il comando **dir** possiede come **input** il **disco** e come **output** lo **schermo**, il primo è espresso in modo esplicito, mentre il secondo è il suo standard input (implicito). Il comando **type** possiede come input il **disco** espresso in modo esplicito e come **standard output** lo **schermo** (implicito).

Analizziamo ad esempio il comando **copy**, che possiede sia **input** e **output** di tipo esplicito. Sia l'input che l'output sono rappresentati dal disco, in quanto riceve un file da disco e lo copia in un'altra destinazione sempre sul disco. Proviamo a reindirizzare l'input di questo comando, portandolo da **disco** (rappresentato da un **file**) a **tastiera**, in modo da prelevare i dati dalla tastiera per essere trasferiti nel file. Per fare questo, essendo il parametro espresso in modo **esplicito**, è sufficiente sostituire il parametro origine con il dispositivo **CON**, come segue:

```
copy CON file.txt
```

In questo caso il sistema attende l'inserimento di dati dalla tastiera, terminanti con il carattere **CTRL Z**.

### Reindirizzamento dello standard input/output

Per effettuare il **reindirizzamento** dello **standard input** o dello **standard output** di un comando utilizziamo i caratteri seguenti:

Operatore di reindirizzamento	Significato
>	Reindirizza l' <b>output implicito (standard output)</b> di un comando
<	Reindirizza l' <b>input implicito (standard input)</b> di un comando
>>	Reindirizza l' <b>output implicito (standard output)</b> di un comando accodando il risultato

Il **reindirizzamento** dell'**output** si effettua con l'operatore **>** che cancella interamente il contenuto del file di destinazione, se questo esiste già. Si può utilizzare anche l'operatore **>>**, con il quale il file di destinazione viene creato, se non esiste, oppure se presente vengono aggiunti i dati alla fine del file destinazione.

Il **reindirizzamento** dell'**input** avviene utilizzando l'operatore **<**, con il quale possiamo inviare un file a un comando utilizzando il flusso dello standard input.

### ESEMPIO Reindirizzare lo standard output

Vogliamo reindirizzare l'output del comando **dir** verso il file **directory.txt**:

```
C:\Users\vo>dir > directory.txt
C:\Users\vo>
```

Se proviamo a visualizzare il contenuto del file, con il comando **type**, ci accorgiamo che contiene l'elenco mostrato dal comando **dir**:

```
C:\Users\vo>type directory.txt
Il volume nell'unità C è OS
Numero di serie del volume: 7437-FA8F

Directory di C:\Users\vo

27/10/2014 21:25 <DIR>
27/10/2014 21:25 <DIR>
03/12/2012 12:58 <DIR> .
07/11/2012 20:41 153 .appleviewer
27/10/2013 20:55 3.487 .gentlproject
29/10/2012 14:52 <DIR> .nbi
26/08/2013 09:49 <DIR> .scuolabook
08/10/2014 19:47 <DIR> .VirtualBox
24/06/2013 20:40 <DIR> Application Data
09/10/2012 13:55 <DIR> bluej
18/03/2014 09:20 <DIR> Contacts
```

```

27/10/2014 20:33 <DIR> Desktop
27/10/2014 21:25 <DIR> directory.txt
08/10/2014 19:55 <DIR> Documents
18/10/2014 22:35 <DIR> Downloads
27/10/2014 20:29 <DIR> Dropbox
26/10/2014 21:43 <DIR> Favorites
27/10/2013 20:55 1.053 ganttproject.log
27/10/2013 20:55 894 java0.log
29/03/2014 22:32 <DIR> Links
29/03/2014 12:05 <DIR> Music
16/10/2014 15:39 <DIR> Pictures

```

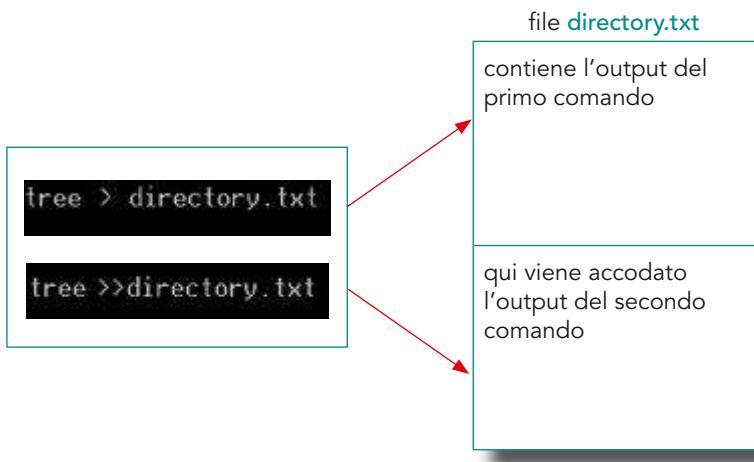
Vediamo adesso invece come accodare dei dati in un file. Come nel caso precedente vogliamo indirizzare l'output, questa volta del comando `tree`, prima della directory `\dev-cpp` e poi della directory `\dev-pas`. Procediamo inizialmente reindirizzando l'output del comando `tree` verso il file `directory.txt`:

```
C:\Users\io>tree \dev-cpp > directory.txt
```

Quindi andiamo ad accodare al file `directory.txt` l'output del comando `tree`, attraverso l'operatore di accodamento `>>`:

```
C:\Users\io>tree \dev-pas >>directory.txt
```

Il file `directory.txt` conterrà a questo punto sia l'elenco del primo comando che quello del secondo:



## ■ I comandi filtro

I **comandi filtro** hanno la caratteristica di non possedere input o output esplicativi. Il tipico comando che si comporta in questo modo è `sort`, che riceve i dati dalla **tastiera**, li **ordina** restituendo il risultato allo **schermo**. Nella sua sintassi non compaiono pertanto altri parametri se non `/+n` che consente di ordinare i dati in base a una determinata colonna:

```
sort /+n
```

Per reindirizzare l'input, prelevando i dati ad esempio da un file (`elenco.txt`), per essere ordinati sulla stampante utilizziamo la seguente sintassi:

```
sort < elenco.txt > PRN
```

In questo caso invece vogliamo utilizzare il comando **sort** per ordinare un file in un altro file.

```
sort < elenco > lista
```

Il comando riceve dallo standard input il file **elenco** e genera attraverso il reindirizzamento dello standard output il file **lista**.

Il comando **find** consente invece di effettuare una ricerca di una stringa all'interno di un file. La sua sintassi è la seguente:

```
find "stringa da cercare" [percorso\file]
```

Ad esempio vogliamo cercare la parola “**Rossi**” nel file **rubrica.txt**, per fare questo utilizziamo il seguente comando:

```
find "Rossi" rubrica.txt
```

In questo caso verranno mostrate tutte le righe che eventualmente conterranno la stringa.

```
C:\>find "Rossi" rubrica.txt
-----
RUBRICA.TXT
Mario Rossi Via Verdini, 34 20100 Milano
C:\>
```



### Prova adesso!

- Utilizzare il reindirizzamento dell'input e dell'output
- Utilizzare il comando sort

- 1 Crea un file di testo (**amici.txt**) che contenga un elenco alfabetico di amici tuoi.
- 2 Ordina il file **amici.txt** in un secondo file chiamato **amici\_o.txt**.
- 3 Crea una rubrica (**rubrica.txt**) inserendovi i dati nel modo seguente:  

Cognome	Nome	Indirizzo	Città	Telefono
Rossi	Mario	Via Verdi,25	Genova	3456789012
Verdi	Luca	Via Paoli, 21	Torino	3331234567
...	...	...	...	...
- 4 Sapendo che ogni dato è stato inserito in una specifica colonna, ordina il file **rubrica.txt** in base alla città in un secondo file chiamato **rubrica\_o.txt**.

## Il pipelining



◀ Il termine indica più elementi collegati da un condotto (**pipe** = tubatura). Più precisamente si riferisce alla possibilità di mettere in contatto comandi in cascata, in modo che il risultato prodotto da uno dei comandi sia l'ingresso di quello immediatamente successivo. ►

Per mettere in contatto lo **standard output** di un comando con lo **standard input** del successivo, si utilizza il **◀ pipelining ▶**. Il simbolo pipe (**|**) viene usato per far ricevere al comando successivo l'output del comando precedente.

**ESEMPIO** *Esempio di pipelining*

In questo esempio vogliamo ordinare un elenco di file di tipo testo (**.txt**) presenti nella directory corrente. Per fare questo utilizziamo la tecnica di **pipelining** tra il comando **dir** e il comando **sort** che segue:

```
C:\>dir *.txt | sort

          0 Directory  405.108.408.320 byte disponibili
          3 File        1.707 byte
Directory di C:\ 
Il volume nell'unità C è OS
Numero di serie del volume: 7437-FA8F
27/10/2014  14:42           14 prova.txt
27/10/2014  21:48           1.522 directory.txt
28/10/2014  21:01           171 rubrica.txt
```

Possono essere collegati anche più comandi, supponiamo di voler cercare il file che contiene la parola “**pro**” nel nome, tra i file con estensione **doc**, della directory **lettere**. Per fare questo dovremmo eseguire i comandi seguenti.

- 1 Ottenerne l'output del comando **dir** nel file **temp** dei file con estensione **.doc**:

```
dir \lettere\*.doc > temp
```

- 2 A questo punto cerchiamo le righe del file **temp** che contengono la parola “**pro**” con il comando **find**:

```
find "pro" temp
```

- 3 Infine cancelliamo il file **temp**, che è stato usato temporaneamente:

```
del temp
```

Lo stesso risultato può essere ottenuto mediante la seguente combinazione di comandi mediante pipelining:

```
dir \lettere\*.txt | find "pro"
```

```
C:\>dir *.doc | find "pro"
28/10/2014  21:27           0 programma.doc
27/10/2014  14:42           14 prova.doc
```

**AREA** *digitale*

- Esempi dei comandi **copy**, **del**, **ren**
- Caratteri jolly nel comando **dir**
- Esempi di pipelining

**AREA** *digitale*

Esercizi per il recupero / Esercizi per l'approfondimento

# ESERCITAZIONI DI LABORATORIO 4

## I FILE BATCH

### ■ I file batch

Il **file batch** sono dei file di testo che contengono una sequenza di **comandi shell**. I comandi contenuti verranno eseguiti in sequenza ogni volta che verrà digitato il nome del file batch al **prompt dei comandi**. I file batch vengono utilizzati ad esempio per evitare di dover ripetere spesso comandi ripetitivi, lunghi e noiosi, oppure ancora complessi e difficili da ricordare.

Si comportano quindi, in realtà, esattamente come un programma che viene **interpretato** ogni volta che lo mandiamo in esecuzione.

Per creare un file batch possiamo usare un editor qualunque, l'importante è assegnare poi al file l'estensione **.bat**.

Ad esempio il file **prova.bat**, contiene i seguenti comandi:

```
dir  
cd \windows  
copy *.* E:\
```

Se al **prompt** dei comandi digitiamo il seguente comando, rappresentato dal nome del **file batch** (**prova**), quindi premiamo **Invio**:

```
C:\>prova
```

Mandiamo in esecuzione il **file batch**, cioè eseguiamo i comandi in esso contenuti. In questo caso otteniamo la copia di tutti i file dalla directory **c:\Windows** alla **root** del disco **E:**.

### ESEMPIO **Spostare file tra due directory**

In questo esempio vogliamo spostare tutti i file dalla directory **\origine** alla directory **\destinazione**, entrambe presenti nell'unità **E:**

```
copy E:\origine\*.* E:\destinazione  
del E:\origine\*.*
```

Come abbiamo visto dopo aver copiato i file li eliminiamo dalla directory di partenza in modo da simulare un effettivo spostamento.

## I parametri sostituibili

Il programma batch può ricevere in **input** alcuni valori particolari provenienti dall'esterno chiamati **parametri sostituibili**. Essi potranno cambiare ogni volta che l'utente decide di avviare il file batch.



Un **parametro** è una stringa che può contenere sia caratteri alfabetici che numeri, introdotte dopo il nome del file batch da eseguire, sempre nella riga di comando. È importante sottolineare che se si introducono più parametri, essi dovranno essere separati tra di loro dallo spazio. ►

All'interno del file batch un parametro sostituibile è rappresentato dal carattere percentuale seguito da un numero progressivo (ad esempio **%1**), che prende il posto del rispettivo parametro passato nella riga di comando. In tal modo possiamo specificare il valore reale delle variabili al momento dell'esecuzione del file. Se ad esempio al file batch **pippo.bat** passiamo i seguenti parametri:

```
pippo casa prova
```

All'interno del file batch i valori che conterranno i parametri sostituibili saranno i seguenti: **%1** conterrà la stringa “**casa**”, mentre **%2** conterrà la stringa “**prova**”.

Il parametro sostituibile **%0** contiene sempre il nome del file batch stesso, così nell'esempio sopra proposto conterrà la stringa “**pippo**”.

### ESEMPIO Spostare file con parametri sostituibili

In questo caso decidiamo di modificare l'esempio precedente, ricevendo i nomi delle due directory come parametri sostituibili. In tal modo il programma potrà essere utilizzato per spostare il contenuto della directory espressa dal primo parametro nella seconda.

```
copy %1\*.* %2
del %1\*.*
```

## I comandi pause ed echo

Il comando **pause** consente di mettere il programma in attesa della pressione di un tasto da parte dell'utente. Tale comando può essere utile per interrompere momentaneamente l'esecuzione del file batch in modo da informare l'utente sul funzionamento del programma, oppure per scandire le operazioni che effettua. Per evitare che mostri il messaggio “Premere un tasto per proseguire...”, possiamo reindirizzare l'output verso un dispositivo inesistente (**NUL**):

```
pause > NUL
```

Il comando **echo** permette di mostrare a video un messaggio rappresentato da una stringa che va posta subito dopo il comando **echo**. La stringa non deve essere posta tra virgolette. Il comando **echo.** permette di andare a capo senza scrivere nulla.

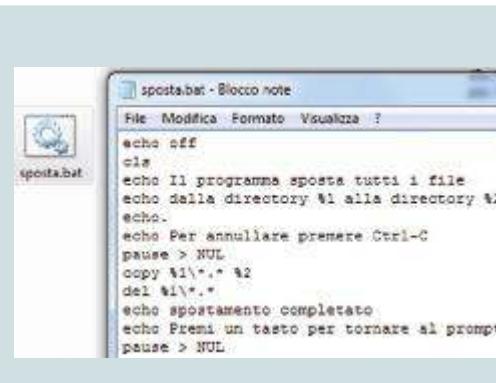
Possiamo disabilitare l'eco dei comandi eseguiti nei file batch mediante l'opzione **off**. Le seguenti righe vengono infatti spesso poste all'inizio di un file batch per disattivare l'echo dei comandi:

```
@echo off
cls
```

### ESEMPIO *Usare echo e pause per migliorare l'interfaccia utente*

In questo caso vogliamo migliorare l'interfaccia utente del nostro programma in modo da informare l'utente sullo scopo del programma. Come possiamo notare all'inizio vengono mostrate due stringhe che informano l'utente sullo scopo del programma: all'interno delle stesse abbiamo collocato anche i parametri sostituibili **%1** e **%2**. Il comando **echo**, viene usato per andare a capo, quindi dopo aver fermato il programma in attesa della pressione di un tasto (**pause > NUL**) viene effettuata lo spostamento dei file. Alla fine del programma un messaggio informa l'utente dell'avvenuta esecuzione.

```
echo off
cls
echo Il programma sposta tutti i file
echo dalla directory %1 alla directory %2
echo.
echo Per annullare premere Ctrl-C
pause > NUL
copy %1\*.* %2
del %1\*.*
echo spostamento completato
echo Premi un tasto per tornare al prompt
pause > NUL
```



Dopo aver scritto il programma con l'editor, lo salviamo:

Vediamo l'esecuzione del programma. Digitiamo il nome del programma (**sposta**), seguito dai due parametri, in questo caso rappresentati da due percorsi relativi (**esercizi**, **esempi**):

```
C:\Users\vio\Desktop>sposta esercizi esempi
```

Dopo aver premuto invio, vediamo il risultato dell'esecuzione del programma:

```
Il programma sposta tutti i file
dalla directory esercizi alla directory esempi

Per annullare premere Ctrl-C
esercizi\nacca.txt
esercizi\asterix.mp3
esercizi\ncopia_es.pas
esercizi\euclide.cpp
esercizi\letter1.docx
esercizi\number.jpg
esercizi\nrime.xls
esercizi\nprova.doc
    8 file copiati.
C:\Users\vio\Desktop\esercizi\*.* Procedere con l'operazione (S/N)? s
spostamento completato
Premi un tasto per tornare al prompt

C:\Users\vio\Desktop>
```



### Prova adesso!

- Applicare i file batch
- Utilizzare i parametri sostituibili

- 1 Modifica l'esempio indicato sopra in modo che l'utente possa decidere di spostare, sempre da una directory a un'altra, solo i file con l'estensione indicata come parametro sostituibile.
- 2 Modifica l'esempio in modo che venga eliminata anche la directory di origine.
- 3 Modifica l'esempio, in modo tale che durante la copia dei file non venga mostrato a video l'elenco degli stessi (utilizzando il reindirizzamento a **NUL** dell'output del comando **copy**).

## Le variabili

Rispetto ai parametri sostituibili le variabili possono avere un nome alfabetico e il loro contenuto può essere stabilito all'interno del programma mediante istruzione di assegnazione. Il nome delle variabili è racchiuso dai due simboli di percentuale (%): ad esempio un nome valido potrebbe essere `%scelta%`. Per assegnare un valore a una variabile si utilizza l'istruzione `set`. Ad esempio per assegnare il valore 5 alla variabile `numero`:

```
set %numero%=5
```

Per leggere un valore da tastiera si utilizza il parametro `/P`. In tal modo il programma si interrompe momentaneamente in attesa di un valore. Il codice seguente legge un valore da tastiera e lo pone nella variabile `scelta`, quindi lo stampa a video nella seconda riga:

```
set /P scelta=Inserire un valore_
echo Hai scelto %scelta%
```

Come possiamo notare la variabile è stata espressa nel comando `set` senza percentuali, mentre quando la utilizziamo, come nella riga seguente, deve essere sempre racchiusa tra percentuali. Il risultato è il seguente:

```
C:\>leggi
C:\>echo off
Inserire un valore_prova
Hai scelto prova
```

### ESEMPIO Utilizza delle variabili

In questo esempio vogliamo modificare l'esempio precedente in modo che le directory origine e destinazione vengano inserite dall'utente durante l'esecuzione del file batch. Per fare questo usiamo due variabili chiamate, rispettivamente, `origine` e `destinazione`:

```
echo off
cls
echo Programma di spostamento file tra 2 directory
set /P origine=Inserisci percorso directory origine
set /P destinazione=Inserisci percorso directory destinazione
echo Per annullare premere Ctrl-C
copy %origine%\*.* %destinazione%
del %origine%\*.*
echo spostamento completato
echo Premi un tasto per tornare al prompt
pause > NUL
```

L'esecuzione è mostrata di seguito:

```
C:\>sposta_var
Programma di spostamento file tra 2 directory
Inserisci percorso directory origineesempi
Inserisci percorso directory destinazioneesercizi
Per annullare premere Ctrl-C
esempi\acca.txt
esempi\asterix.mp3
esempi\copia_es.pas
esempi\euclide.cpp
esempi\letter1.docx
esempi\lumber.jpg
esempi\primo.xls
esempi\prova.doc
          8 file copiati.
C:\Users\vio\Desktop\esempi\*.* Procedere con l'operazione (S/N)? s
spostamento completato.
Premi un tasto per tornare al prompt
```

## ■ La selezione

La **selezione** consente di far proseguire l'esecuzione del programma in modo diverso a seconda dell'esito di una **condizione**. La selezione avviene con il comando **if**, che può verificare le seguenti condizioni:

- ▶ se una variabile o un parametro **contiene** una stringa;
- ▶ se **esiste** un file o una directory;
- ▶ se il codice di uscita di un programma è **maggior o uguale** a un valore.

Nel primo caso la sintassi è la seguente:

```
if [not] [stringa | variabile | parametro] == stringa
```

La sintassi prevede anche la **negazione** per la condizione di non uguaglianza mediante l'operatore **not** posto davanti alla condizione, ad esempio la selezione salta all'etichetta errore se la variabile a contiene un valore diverso da 1:

```
if not %a% == 1 goto errore
```

Sostanzialmente la sintassi può essere la seguente:

```
if variabile/parametro == stringa goto etichetta
```

Le parole chiave sono **if**, il simbolo di doppio uguale (**==**) e il **goto**. Ad esempio: se la variabile **scelta** contiene la stringa “esci” il programma deve proseguire a partire dall'etichetta **fine**.

```
if %scelta% == esci goto fine
...
:fine
```

I file batch utilizzano la programmazione con i goto, nonostante sia assolutamente deprecata nella programmazione strutturata. Tuttavia ricordiamo di non applicare questa tecnica anche alla programmazione tradizionale.

### ESEMPIO **Gestione rubrica**

In questo esempio utilizzeremo le variabili per scegliere l'operazione da effettuare su di una rubrica mediante un menu. Il menu è il seguente:

1. Apri la rubrica (per modifica, aggiunta, eliminazione)
2. Visualizza la rubrica
3. Ordina tramite il cognome
4. Ordina tramite città
5. Ricerca un nominativo
6. Uscita

Il file (file **rubrica.txt**) di testo che contiene i dati della rubrica, sul quale opera il file batch, possiede cinque diverse colonne, il Cognome (1), il Nome (15), l'indirizzo (30), la città (45) e il telefono (60):

Cognome	Nome	Indirizzo	Città	Telefono
Rossi	Mario	Via Fontana,3	Milano	0234567890
Verdi	Alberto	Via Manzoni, 34	Livorno	0586123456
...	...	...	...	...

Il codice del file principale (**menu.bat**) è il seguente:

```
:inizio
echo off
cls
echo -----Menu di scelta-----
echo 1. Apri la rubrica (per modifica, aggiunta, eliminazione)
echo 2. Visualizza la rubrica
echo 3. Ordina tramite il cognome
echo 4. Ordina tramite città
echo 5. Ricerca un nominativo
echo 6. Uscita
echo -----
set /P scelta=Effettua la scelta_
if %scelta%==1 goto apri
if %scelta%==2 goto visual
if %scelta%==3 goto ord_cognome
if %scelta%==4 goto ord_citta
if %scelta%==5 goto ricerca
if %scelta%==6 goto esci
goto errore
:apri
notepad rubrica.txt
pause > NUL
goto inizio
:visual
type rubrica.txt | more
pause > NUL
goto inizio
:ord_cognome
sort < rubrica.txt | more
pause > NUL
goto inizio
:ord_citta
sort /+45 < rubrica.txt | more
pause > NUL
goto inizio
:ricerca
set /P cerca=Inserisci il nominativo da cercare
find "%cerca%" rubrica.txt
pause > NUL
goto inizio
:errore
echo Scelta non valida!
pause > NUL
goto inizio
pause
:esci
echo Grazie per aver usato il programma
pause
cls
```

Come possiamo notare il codice iniziale mostra il menu, quindi richiede all'utente la scelta nella variabile omonima. Mediante alcune istruzioni di selezione (`if`) vengono verificate le scelte effettuate. Se il valore inserito dall'utente nella variabile scelta è diverso da tutti quelli presenti, quindi non compreso tra 1 e 6, viene eseguita l'istruzione che richiama l'etichetta `errore` (`goto errore`). In questa etichetta viene segnalato un errore, quindi viene richiamato il programma dall'inizio (`goto inizio`), dopo aver fatto premere un tasto all'utente. Vediamo lo sviluppo delle diverse parti del programma, ciascuna identificata da una diversa etichetta.

1. La sezione di etichetta `:apri` richiama un editor e apre contestualmente il file `rubrica.txt` (`notepad rubrica.txt`).
2. La sezione di etichetta `:visual`, mostra a video tutto il contenuto del file `rubrica.txt` a pagine attraverso il comando `type rubrica.txt | more`
3. La sezione di etichetta `:ord_cognome`, ordina il file `rubrica.txt` mediante la prima colonna (quella del cognome) e mostra a video a pagine il risultato mediante il comando `sort < rubrica.txt | more`.
4. La sezione di etichetta `:ord_citta`, ordina il file `rubrica.txt` mediante la colonna 45 (quella della città) e mostra a video a pagine il risultato mediante il comando `sort /+45 < rubrica.txt | more`.
5. La sezione di etichetta `:ricerca`, chiede all'utente il nominativo da cercare attraverso il comando `set /P cerca=Inserisci il nominativo da cercare`. La variabile `cerca` contiene quindi la stringa utile alla ricerca che viene utilizzata nel comando `find "%cerca%" rubrica.txt` per ottenere le righe del risultato.
6. Infine la sezione di etichetta `:esci`, esce dal programma mostrando un messaggio.

Tutte le etichette, a esclusione dell'etichetta `:esci`, possiedono le stesse istruzioni di attesa di un tasto premuto (`pause>NUL`) e di salto all'inizio (`goto inizio`).



### La verifica di esistenza di una directory

Per verificare se una directory oppure un file **esistono** utilizziamo il comando `if exist`. La sintassi del comando è la seguente:

```
if [not] exist [percorso\file] goto etichetta
```

Questo comando può essere usato, ad esempio, per “testare” l'esistenza di un file, il cui nome è contenuto in una variabile, prima di effettuare una operazione su di esso, come una copia oppure una cancellazione:

```
:inizio
set /P file=Inserisci il nome del file da copiare_
...
if exist %file% goto copia
goto errore
...
:copia
copy %file% \destinazione
:errore
echo Attenzione il file è inesistente!
goto inizio
```

Come possiamo notare l'istruzione seguente al comando `if` viene eseguita solo quando la condizione dà esito negativo.

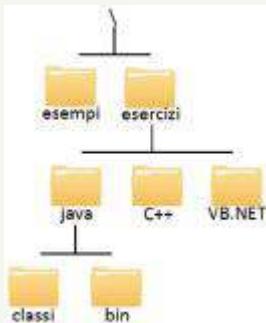
#### ESEMPIO *Programma di spostamento con verifica sull'esistenza*

In questo caso riprendiamo l'esempio dello spostamento di file da una directory a un'altra. Vogliamo fare in modo che se l'utente ha immesso delle directory non esistenti appaia un messaggio di errore e non venga eseguito il comando. Dopo aver letto le due variabili `origine` e `destinazione`, il programma verifica mediante due selezioni (`if`) che le variabili non siano vuote (`if var=="" goto err_st`). In caso contrario viene richiamata la sezione `err_st` dove viene visualizzato un messaggio di errore. Successivamente viene verificata l'esistenza delle due directory mediante due controlli di selezione (`if not exist var goto err_file`). Nella sezione di etichetta `err_file` viene inviato a video un messaggio di errore.

```
echo off
cls
echo Programma di spostamento file tra 2 directory
set /P origine=Inserisci percorso directory origine
set /P destinazione=Inserisci percorso directory destinazione
if %origine%=="" goto err_st
if %destinazione%=="" goto err_st
if not exist %origine% goto err_file
if not exist %destinazione% goto err_file
copy %origine%\*.* %destinazione%
del %origine%\*.*
echo spostamento completato
goto fine
:err_st
echo Devi inserire un nome di directory valido
goto fine
:err_file
echo La directory non esiste!
:fine
echo Premi un tasto per tornare al prompt
pause > NUL
```



## Prova adesso!



- Utilizzare la selezione
- Applicare le variabili
- Utilizzare la condizione di esistenza di una directory

- 1 Realizza un file batch che crea un albero di directory se non è stato creato, se invece è già presente lo deve eliminare. L'albero da creare è il mostrato in figura.
- 2 All'interno della cartella **java** deve anche creare un file che contiene l'output della root del sistema.

## ■ Il comando for

Il comando **for** permette di effettuare un ciclo in cui ripetere un comando. La sintassi è la seguente:

```
for %%var in (par1 par2... par"n") do comando
```

Durante il ciclo **for** viene eseguito il comando presente dopo la parola chiave **do**, per un numero di volte pari al numero di parametri posti tra parentesi dopo la parola chiave **in**. A ogni ciclo, la variabile **%var** che viene usato nel comando, viene sostituita dal parametro posto tra parentesi.

### ESEMPIO *Stampa i file di estensioni diverse*

In questo esempio vogliamo accodare il contenuto di tutti i file di testo con estensione **txt**, **pdf** e **doc** nel file tutti della directory corrente. Ciascun file con estensione indicata nella parentesi, viene sostituito alla variabile **%n** finché.

```
for %%n in (*.txt *.pdf *.doc) do type %%n >> tutti
```

### ESEMPIO *Sposta i file*

Si desidera realizzare un file batch che sposti tutti i file specificati sulla linea di comando dalla directory **origine** alla directory **destinazione** dell'unità **E**. In questo caso i file sono rappresentati dai parametri sostituibili passati al file batch. Si assume inoltre che i file siano al massimo cinque.

```
for %%n in (%1 %2 %3 %4 %5 ) do copy E:\origine\%%n E:\destinazione
for %%n in (%1 %2 %3 %4 %5 ) do del E:\origine\%%n
```

## AREA digitale



Esercizi per il recupero / Esercizi per l'approfondimento

# 5

# Fasi e modelli di gestione di un ciclo di sviluppo

- L1 **Modelli classici di sviluppo di sistemi informatici**
- L2 **Un nuovo modello di sviluppo: OOP**
- L3 **Documentazione di un progetto**

## Esercitazioni di laboratorio

① I diagrammi UML con ArgoUML; ② Il Model-Oriented Programming con Umple

### Conoscenze

- Comprendere le necessità di una metodologia per lo sviluppo di sistemi informatici
- Conoscere gli elementi fondamentali dell'ingegneria del software
- Conoscere gli elementi teorici della progettazione a oggetti (OOP)
- Capire l'utilizzo delle schede CRC per l'identificazione di classi
- Conoscere una metodologia di documentazione (UML)

### Competenze

- Individuare e descrivere il problema complesso
- Usare la progettazione orientata agli oggetti per programmi complessi
- 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

### Abilità

- Scegliere le metodologie e le tecniche adeguate alle diverse situazioni
- Applicare il concetto di astrazione per modellare le classi
- Utilizzare ArgoUML per documentare un progetto

## AREA *digitale*



► Esercizi



- Il processo unificato
- Elenco dei documenti di un progetto
- La qualità del software con la norma ISO 9000
- L'UML come metamodello
- Articolo di Timothy Lethbridge su MOP & Umple
- Esercizi per il recupero
- Esercizi per l'approfondimento



Soluzioni (prova adesso, esercizi, verifiche)  
Puoi scaricare il file anche da [hoepliscuola.it](http://hoepliscuola.it)

# Modelli classici di sviluppo di sistemi informatici

In questa lezione impareremo...

- ▶ a individuare e descrivere un problema
- ▶ a costruire la soluzione di un problema
- ▶ ad affrontare correttamente la ricerca della soluzione
- ▶ a scegliere le metodologie e le tecniche appropriate 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 **Milano EXPO 2015**, è indispensabile utilizzare un approccio rigoroso in ogni fase della **realizzazione**, dalla **pianificazione** alla **progettazione** e infine al “collaudo sul campo”.

La produzione del software non può essere affidata all'**improvvisazione**: sapere scrivere algoritmi e programmi non garantisce di per sé l'avere un software di **buona qualità**.

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, soprattutto adottando tecniche specifiche che facilitino ogni fase di lavorazione del **prodotto software**.



### PRODOTTO SOFTWARE

Insieme di tutti gli artefatti che permettono l'utilizzo di un programma da parte di un utente: codice, documentazione, prodotti intermedi quali casi di test, manuali tecnici ecc.

In questa lezione 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.



### 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.

I primi programmatori degli anni Sessanta del secolo scorso spesso condividevano la stessa formazione scientifica degli utilizzatori che, per avere a che fare con gli elaboratori elettronici, dovevano necessariamente avere avuto una formazione tecnica: i programmatori avevano una certa familiarità con i problemi degli utenti dato che riguardavano essenzialmente situazioni di matematica o statistica e spesso erano gli utenti stessi che “si producevano i programmi”!

Il software realizzato in quegli anni era caratterizzato dal fatto di essere “un'applicazione prodotta *ad hoc*” per risolvere un particolare problema e che venisse “dimenticato” una volta soddisfatta quella particolare necessità.

Il programmatore degli anni Sessanta era un artista vero e proprio, nel senso che volta per volta si lasciava prendere dall'inventiva del momento per realizzare il programma a lui richiesto, senza adottare particolari tecniche di sviluppo o utilizzare strumenti adeguati che, in quegli anni, scarseggiavano se non erano del tutto mancanti.

Il secondo periodo di sviluppo è iniziato con l'era delle applicazioni gestionali, cioè con l'utilizzo dell'informatica per i settori commerciale e amministrativo: una nuova tipologia di utenti “si avvicina” all'utilizzo del software, generalmente con formazioni culturali diverse da quella dei programmatori e quindi quest'ultimi devono affrontare nuove problematiche, quali comunicare con persone che “non parlano la loro lingua”.

Devono anche imparare ad affrontare nuovi problemi, lontani dalla loro formazione scientifica: il programmatore deve quindi acquisire nuove nozioni prima di poter scrivere il

programma, necessarie proprio per avere gli elementi della **conoscenza del problema**, e solo successivamente può iniziare il lavoro di progetto.

Il **primo problema** è proprio quello di "capire il problema"!

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!) si deve progettare la soluzione che risolve il problema.

Programmare non è quindi un'operazione semplice!

Un aspetto che inoltre non può essere trascurato è anche quello economico: le applicazioni richieste hanno un peso economico sempre maggiore e assumono un ruolo di importanza strategica.

Il programmatore non può più più "operare da solo" ma deve sviluppare i prodotti software collaborando con altri programmatore: nascono molte imprese autonome appositamente dedicate alla realizzazione di programmi (**software houses**) oppure, in alternativa, vengono formate sezioni nelle grosse aziende destinate a tali scopi (**CED** aziendali per l'elaborazione dati o di sviluppo del software).

Ma anche in questa fase, nonostante ci sia aggregazione, prevale l'approccio individualistico alla programmazione soprattutto per la mancanza di strumenti teorici e di pratiche standardizzate: la principale tecnica è quella chiamata **seniority (anzianità)**, cioè il frutto dell'esperienza "maturata sul campo", ottenuta con il perfezionamento delle metodologie classiche.

A essa si aggiungono gli "strumenti personali" propri (e indispensabili) dei quali ogni programmatore è dotato: l'esperienza, la fantasia, l'intuito e l'ingegno.

L'inadeguatezza degli strumenti teorici e metodologici nella fase artigianale della produzione di software fece sentire i propri effetti nella famosa **crisi del software**, fra gli anni Sessanta e Settanta: la produzione del software non riusciva a tenere il passo con le richieste degli utenti e con i progressi dello hardware, aumentavano i **costi di sviluppo**, i **ritardi nelle consegne**, e i **malfunzionamenti**, con un conseguente aumento dei costi di manutenzione.

## ■ Ingegneria del software e ciclo di vita

La crisi del software portò alla acquisizione della consapevolezza che il software doveva passare da un "livello artigianale" a un "metodo industriale": era cioè necessario usare nella produzione di software lo stesso approccio che si usa nelle industrie mature, come l'industria meccanica o l'industria elettronica.

Nella conferenza di **Garmisch** del 1968 venne coniato il termine di “**ingegneria del software**”.

L'**ingegneria del software** nacque per formalizzare e definire le tecniche e le strategie finalizzate all'ottimizzazione e al miglioramento della produzione dei programmi, ma ancora oggi la transizione da artigianato a industria non è del tutto completata dato che molte imprese del settore sono ancora legate alla vecchia impostazione “casalinga”.

La maggior parte dei produttori di software ha adottato **processi di sviluppo strutturati** e metodologie di carattere ingegneristico.

### Processo software

Con ▶ **processo software** ▷ si intendono l'insieme organizzato di pratiche (attività) che sovrintendono alla costruzione del prodotto da parte del team di sviluppo utilizzando **metodi, tecniche, metodologie e strumenti**.

◀ **Standard IEEE 610.12-1990 Software development process:** The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. ▷



La produzione del software dagli anni Sessanta inizia una sua “rivoluzione” attraverso l'introduzione di nuove tecniche e metodologie in grado di affrontare e risolvere i problemi aventi le seguenti finalità:

- ▶ la **formalizzazione** dei metodi di *sviluppo* e di *documentazione* del software;
- ▶ la **realizzazione** di prodotti software di *buona qualità* e a *costi contenuti*;
- ▶ la **riduzione** della *complessità* della tecnologia;
- ▶ la **facilità** di utilizzo dei programmi da parte dell'utente.

Il processo di sviluppo del software viene scomposto in fasi e nascono un insieme di **modelli** che organizzano in fasi successive le diverse attività connesse alla “produzione di un programma” secondo uno schema di riferimento (il **ciclo di vita del software**).

### Ciclo di vita del software

L'**ingegneria del software** 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.

◀ **Standard IEEE 12207.0-1996 Software life cycle CSV:** The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. ▷



**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 Settanta: 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.

Il numero, il contenuto e la denominazione delle fasi può variare da un'organizzazione all'altra, e da un progetto all'altro. Oggi uno schema di riferimento è il seguente, dove le fasi sono scomposte in sottofasi:

- **studio di fattibilità** – pianificazione del sistema
- **analisi e specifica dei requisiti**, suddivisa in:
  - analisi (definizione e specifica) dei requisiti dell'utente
  - specifica dei requisiti del software
- **progetto**, suddiviso in:
  - progetto architetturale
  - progetto in dettaglio
- **codifica singole unità software**
- **programmazione e test di unità**
- **installazione**
  - integrazione
  - test di sistema
- **manutenzione**

Contemporaneamente a queste fasi, e nel corso di tutto il processo, si svolgono anche queste attività di supporto come il controllo di qualità e la stesura della documentazione.

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 e 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 riassunto nella tabella a lato. ►

Pianificazione del sistema Analisi, scomposizione sottosistemi Formalizzazione	Come il modello “a cascata”
Sottosistema 1	
Codifica di un sottosistema 1	
Test del sottosistema 1	
Installazione	
Sottosistema 2	
Codifica di un sottosistema 2	
Test del sottosistema 2	
Installazione	
Integrazione con altri sottosistemi	Modello a incrementi
...	
Sottosistema N	
Codifica di un sottosistema 1	
Test del sottosistema 2	
Installazione 3	
Integrazione con altri sottosistemi	
Manutenzione	Come il modello “a cascata”

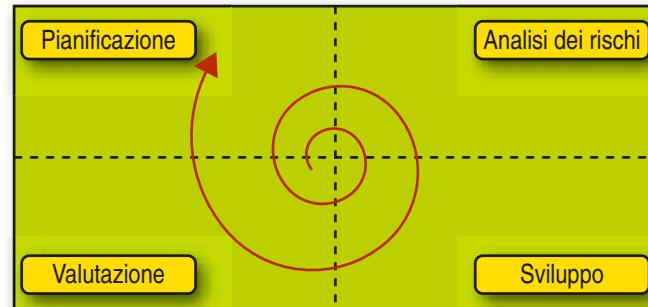
## ■ Modello a spirale

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, scomponete 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 problematiche 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.

## ■ Sviluppo “agile” o iterativo incrementale

Nel corso degli anni Novanta 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 in quanto "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: è quindi necessario un procedimento iterativo e le metodologie (in contrapposizione alle metodologie pesanti) si basano proprio su fasi di progettazione, sviluppo e test molto più brevi.

Le tre principali caratteristiche delle metodologie agili sono:

- **iteratività 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).

Il gruppo di sviluppo deve essere piccolo, molto affiatato, con elevata esperienza e motivato e deve interagire continuamente col cliente e l'utente finale: le richieste di cambiamento del progetto, anche in fase avanzata di sviluppo, vengono considerate positivamente e un **team agile** è un team in grado di rispondere in modo appropriato ai cambiamenti.

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 nella **programmazione estremizzata** (*Extreme Programming*) il **modello agile** più famoso dove la semplicità viene posta come elemento fondamentale eliminando la maggior parte dei vincoli formali di una metodologia di sviluppo tradizionale.

**Beck** indica le seguenti regole pratiche come basi del modello **XP** (*Extreme Programming*):

REGOLA	DESCRIZIONE
Pianificazione realistica	I clienti devono prendere le decisioni sulla funzionalità, i programmatore 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 programmatore 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 programmatore sia i clienti devono preparare casi di prova. Il sistema deve essere collaudato continuamente
Riprogettazione	I programmatore devono continuamente ristrutturare il sistema per migliorare il codice ed eliminare parti duplicate
Programmazione a coppie	I programmatore devono lavorare a coppie e ciascuna coppia deve scrivere codice su un unico calcolatore
Proprietà collettiva	Tutti i programmatore 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, riempiedoli 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 programmatore 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 **Program-**

**mazione estremizzata** consiste nella sinergia tra queste regole pratiche, cioè che la somma è maggiore degli addendi.

Alcune di queste verranno riprese nel processo unificato UP proposto nel 1999 da **Jacobson, Booch e Rumbaugh**.



## ■ 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 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.

I **modelli agili** sono da preferire quando si deve garantire la consegna di un prodotto in tempi estremamente rapidi ma sono difficilmente applicabili in sistemi di grandi dimensioni in quanto è difficile avere una corretta documentazione di tutte le parti realizzate e da realizzare.

## Verifichiamo le conoscenze

### 1. Risposta multipla

**1 Cos'è la seniority?**

- a. L'età dei programmatori
- b. L'età degli analisti
- c. L'esperienza di progettazione
- d. L'uso delle metodologie

**2 La manutenzione ha i seguenti aspetti (indicare quello errato):**

- a. l'aspetto correttivo
- b. l'aspetto migliorativo
- c. l'aspetto qualitativo
- d. l'aspetto adattativo

**3 Quale tra le seguenti fasi non rientra nel modello a prototipazione rapida?**

- a. Raccolta e rifinitura dei requisiti
- b. Progetto rapido
- c. Costruzione del prototipo

- d. Valutazione del prototipo
- e. Rifinitura del prototipo
- f. Definizione delle interfacce
- g. Ingegnerizzazione del prodotto

**4 Quale tra le seguenti fasi non rientra nel modello a spirale?**

- a. Analisi dei rischi
- b. Analisi dei benefici
- c. Sviluppo
- d. Valutazione
- e. Pianificazione

**5 I rischi nel modello a spirale sono riferiti a:**

- a. l'aspetto economico
- b. l'aspetto temporale
- c. l'aspetto giuridico
- d. l'aspetto tecnologico

### 2. Ordinamento

**1 Ordina cronologicamente secondo il modello a cascata.**

- a. Codifica (realizzazione) .....
- b. Progettazione .....
- c. Pianificazione del sistema .....
- d. Test e debug .....
- e. Analisi .....
- f. Manutenzione .....
- g. Installazione, verifica e collaudo .....

### 3. Vero o falso

- 1 Nel modello a cascata ogni singola attività viene completata prima del passaggio alla successiva.
- 2 Nel modello a cascata è prevista la fase di revisione del progetto.
- 3 La manutenzione è la fase permanente di supporto al sistema dopo la consegna all'utente.
- 4 Il processo a prototipazione rapida di sviluppo inizia con la fase di raccolta dei requisiti.
- 5 Il processo a prototipazione rapida di sviluppo finisce con la fase di rifinitura del prodotto.
- 6 Nel modello incrementale ogni fase è un incremento della precedente.
- 7 L'analisi dei rischi è fondamentale nel modello incrementale.
- 8 Tra i rischi non è da trascurare la capacità del gestore del progetto.
- 9 Il modello a spirale è estremamente recente, definito nel 1998.
- 10 Nel modello a spirale è presente una fase di valutazione.
- 11 Ogni modello di sviluppo migliora il modello precedente.
- 12 Le metodologie agili sono di tipo adattativo e non predittivo.



# Un nuovo modello di sviluppo: OOP

In questa lezione impareremo...

- ▶ il concetto di programmazione di sistema
- ▶ la motivazione della crisi del software
- ▶ il concetto di classe, oggetto, encapsulamento, 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 Novanta.

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**”.

Ricon sideriamo 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 Ottanta 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:

- |  |  |
|--|--|
| <b>1</b> ben organizzato<br><b>2</b> modulare<br><b>3</b> protetto<br><b>4</b> riusabile<br><b>5</b> riconfigurabile | <b>6</b> flessibile<br><b>7</b> documentato<br><b>8</b> incrementalmente estendibile<br><b>9</b> a componenti<br><b>10</b> ... |
|--|--|

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

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”.

Anche l'evoluzione tecnologica ha portato a esigenze di nuove tecniche e strumenti di programmazione: si pensi agli smartphone, ai pocket pc, ai tablet e alla loro interfaccia con l'utente, diversa della tradizionale tastiera a 102 tasti e dal monitor passivo.

#### ESEMPIO

I linguaggi devono adeguarsi alla necessità di una nuova modalità di comunicazione, dove l'**interfaccia grafica** e modalità di I/O come gli **schermi touch** diventano indispensabili: 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 1](#) e [SIMULA 67](#), sviluppati da [Ole-Johan Dahl](#) e [Kristen Nygaard](#) all’inizio degli anni Sessanta, 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 Settanta 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](#), [C++](#) 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 Ottanta ha rappresentato un serio problema, al punto di mettere in crisi più volte il “sistema informatico” (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 programmati”**, 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 programmatorei:** se assegnate mille programmatorei 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 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 trascorso 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 subito 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 programmatorei 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, encapsulamento, 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.

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 di 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.

## ■ Conclusioni: che cos’è la programmazione a oggetti

Si è detto che la programmazione **OOP** nasce alla fine degli anni Ottanta 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**.

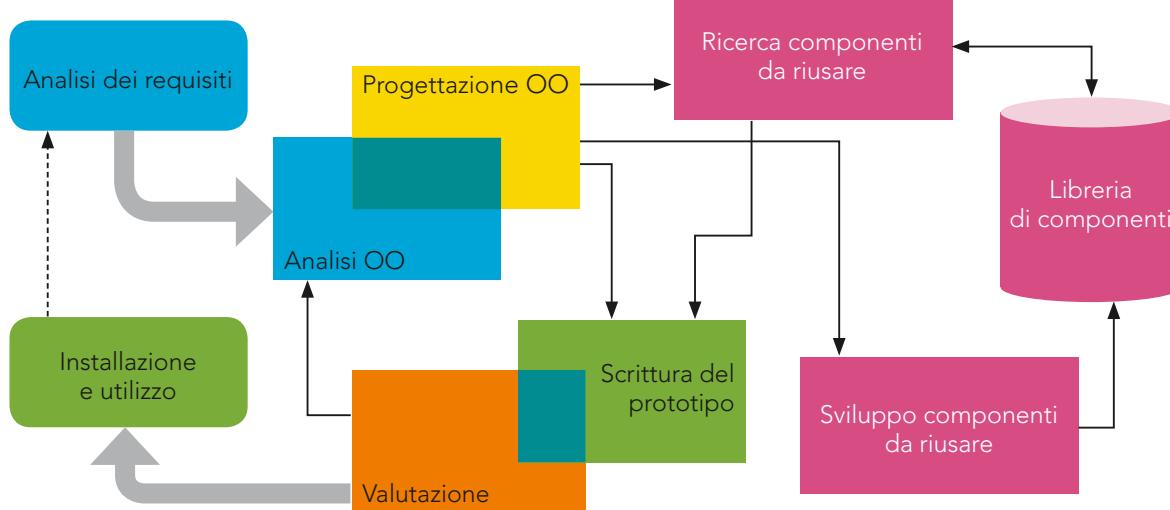
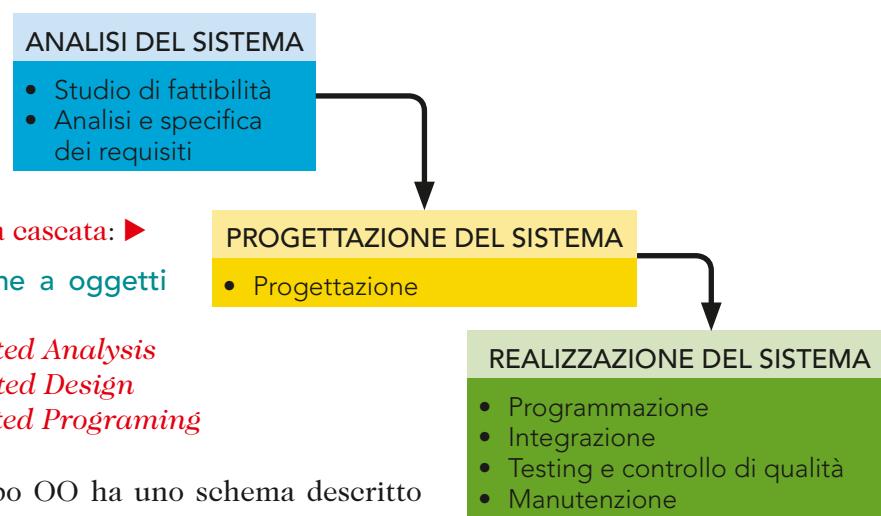
Un'ulteriore caratteristica della **OOP** è quella che prende il nome di **polimorfismo**: si tratta della possibilità di applicare lo stesso operatore a classi diverse, cioè mandare un messaggio con un "medesimo significato" a classi diverse che però lo interpretano e lo eseguono in maniera differente, compatibile con la loro struttura.

Le tecnologie **OO** permeano tutto il **ciclo di sviluppo** del software e possiamo individuare tre distinte macrofasi, in analogia col **modello a cascata**:

Nella **programmazione a oggetti** prendono il nome di:

- ▶ **OOA: Object Oriented Analysis**
- ▶ **OOD: Object Oriented Design**
- ▶ **OOP: Object-Oriented Programming**

Il processo di sviluppo OO ha uno schema descritto nella seguente figura:



Si noti come è stato evidenziato il fatto che un obiettivo è quello di "cercare di riutilizzare" componenti già pronti, disponibili nelle librerie e quindi già "funzionanti".

Di norma vengono creati e raffinati prototipi funzionanti del sistema o di sue parti, secondo l'approccio **incrementale-iterativo**, cioè:

- si individuano sottoparti relativamente autonome;
- si realizza il prototipo di una di esse;
- si continua con altre parti;
- si aumenta progressivamente l'estensione e il dettaglio dei prototipi, tenendo conto delle altre parti interagenti.



### LINGUAGGIO DI MODELLAZIONE

Un **linguaggio di modellazione** fornisce le primitive a cui ricondurre la realtà in esame e permette di esprimere le **entità** che compongono un sistema complesso, le loro **caratteristiche** e le **relazioni** che le collegano.

I progettisti del modello a oggetti utilizzano quasi esclusivamente un unico linguaggio di modellazione di tipo grafico, l'**UML** (*Unified Modeling Language*), sviluppato verso la metà degli anni Novanta da **Grady Booch, James Rumbaugh, Ivar Jacobson**.

L'**UML** è un linguaggio **semiformale** e **grafico** che utilizza specifici diagrammi per specificare, visualizzare, costruire e documentare gli artefatti di un sistema software: non è solo un linguaggio per la modellazione, ma un linguaggio per la modellazione **orientata agli oggetti**.

Quindi include sia l'analisi che la progettazione orientata agli oggetti, cioè sia la **OOA** e **OOD**:

- **analisi**: capire **cosa** deve fare il sistema, senza occuparsi dei dettagli implementativi;
- **progettazione**: capire **come** il sistema raggiunge il suo scopo, come viene implementato.

**UML** offre strumenti di modellazione **OO** in entrambi gli ambiti; frammenti differenti di **UML** sono impiegati in diverse fasi del processo di sviluppo, come ad esempio nel progetto dei database per la descrizione dei diagrammi **E-R** (sarà descritto nella prossima lezione).

## Verifichiamo le conoscenze

### 1. Risposta multipla

**1 L'acronimo OOP sta a indicare:**

- a. Object Oggettive Programming
- b. Oggettive Object Processing
- c. Object Oriented Programming
- d. Oriented Object Processing

- a. dimensionale
- b. gestionale
- c. qualitativa
- d. quantitativa

**2 Quale tra le seguenti non è una caratteristica di un sistema software?**

- |                      |                      |
|----------------------|----------------------|
| a. È ben organizzato | e. È riusabile       |
| b. È modulare        | f. È riconfigurabile |
| c. È protetto        | g. È flessibile      |
| d. È semplice        |                      |

**3 Quale tra i seguenti non è un paradigma di programmazione?**

- |               |               |
|---------------|---------------|
| a. Imperativo | c. Matematico |
| b. Funzionale | d. Logico     |

**6 Quale tra le seguenti affermazioni è errata?**

- a. I metodi corrispondono ai programmi
- b. Gli attributi corrispondono alle variabili
- c. Un oggetto è una istanza di una classe
- d. Una classe è "un mattone software"

**4 Quale fu il primo linguaggio a oggetti?**

- |             |              |
|-------------|--------------|
| a. Ada      | d. C++       |
| b. Java     | e. SmallTalk |
| c. Simula I |              |

**7 I metodi possono essere (3 risposte):**

- |                |                   |
|----------------|-------------------|
| a. costruttori | c. modificatori   |
| b. distruttori | d. visualizzatori |

**8 L'incapsulamento consiste:**

- a. nell'astrarre gli oggetti
- b. nel fare in modo che i dettagli restino invisibili all'esterno
- c. nel propagare automaticamente attributi comuni
- d. nel realizzare il divide et impera

**9 Quale tra i seguenti non è un concetto OOP?**

- |                   |                 |
|-------------------|-----------------|
| a. Incapsulamento | c. Polimorfismo |
| b. Automatismo    | d. Ereditarietà |

**5 La crisi del software non fu di natura:**

### 2. Vero o falso

**1 Il primo linguaggio a oggetti fu sviluppato negli anni Novanta.**

V F

**2 Con il termine legacy si intende una applicazione OOP.**

V F

**3 L'astrazione ha un ruolo fondamentale nelle OOP.**

V F

**4 SmallTalk fu introdotto negli anni Settanta ed è il primo linguaggio "puro".**

V F

**5 La classe è un insieme di oggetti.**

V F

**6 L'oggetto è una 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

# Documentazione di un progetto

In questa lezione impareremo...

- ▶ a comprendere l'importanza della documentazione
- ▶ 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

## ■ La documentazione del software

La realizzazione di un progetto software non può prescindere da una gestione accurata della **documentazione**: sin dalle fasi iniziali, cioè della contrattazione fino ai documenti di collaudo passando per i vari momenti di stato di aggiornamento dei lavori, ogni fase e ogni attività deve essere documentata.

Una **documentazione** precisa consente di schematizzare le parti, di evidenziare le risorse disponibili, di precisare le tecniche che si sono adottate, di fungere da supporto completo per una comunicazione proficua con tutte le figure, utente, analista, programmatore ecc. che concorrono alla realizzazione del software.

Il responsabile del progetto (**Project Manager**) e il **team di lavoro** devono poter accedere e aggiornare qualsiasi **documento** inerente allo sviluppo e alla gestione ordinaria dei progetti, dai più semplici ai più complessi, durante la realizzazione di ogni fase dello sviluppo: si dovrà quindi predisporre una documentazione adeguata all'utente a cui è destinata, dalla stesura del materiale riassuntivo sulla logica del progetto fino alla produzione dei manuali operativi per gli utenti finali.

Molto spesso quando si propone la scrittura di un **documento** si sollevano alcune obiezioni:

- ▶ la **documentazione** è inutile perché non la legge nessuno;
- ▶ il tempo necessario per scrivere (e leggere) la **documentazione** può essere “meglio utilizzato” per prove e sviluppo di software (dato che, di norma, si è sempre “in ritardo”).

L'esperienza ha dimostrato che quanto più precisa sarà la **documentazione di progetto** tanto maggiore sarà la **qualità di un prodotto** e quindi la sua vendibilità.

Una buona **documentazione** non è garanzia di un buon progetto ma è difficile realizzare un buon progetto senza o con una cattiva **documentazione**.

## AREA digitale



Elenco dei documenti di un progetto

L'importanza della **documentazione** che correda un progetto software diventa ancora maggiore nel momento in cui ci si prefigge l'obiettivo di rendere **portabile** e **riusabile** il software prodotto.

La **OOP** ha proprio tra i suoi primari obiettivi quello di garantire un livello **elevato di riuso** delle componenti software e semplificare il processo di **porting** del software su piattaforme differenti.

La fase di progetto produce alcuni **documenti** che descrivono l'architettura del sistema: parte di tali **documenti** sono scritti in linguaggio naturale seguendo una strutturazione standard adottata per tutto il progetto e in parte mediante notazioni di progetto **testuali** e **grafiche**, utilizzando metodologie standard come le schede **CRC** e i diagrammi **UML** che descriveremo in questa lezione.

## ■ Documentare un progetto a OOP

Nel **modello orientato agli oggetti** la documentazione grafica avviene a diversi livelli che sinteticamente possiamo riassumere in:

- 1 individuazione delle classi;
- 2 determinazione delle responsabilità di ogni classe;
- 3 descrizione delle relazioni tra le classi individuate.

L'identificazione delle classi viene effettuata mediante apposite schede, le **CRC** (**C**lasse, **R**esponsabilità, **C**ollaboratori) e in ciascuna di esse si descrive una singola classe, le sue responsabilità, cioè le azioni che svolge, e i suoi collaboratori, cioè le classi che interagiscono con essa in ogni singola attività.

Per completare le classi spesso si “parte a rovescio”, cioè per ogni azione e/o compito si individua e/o definisce una classe responsabile e la si annota sulla scheda indicando anche le altre classi necessarie per assolvere la responsabilità (collaboratori).



◀ Con **modellazione** si intende la “tipica attività ingegneristica” che consiste nel costruire un modello del manufatto che si vuole realizzare, dove con modello si intende una rappresentazione di un sistema reale che ne cattura gli aspetti di interesse. ►

Dato che la documentazione di un progetto **object oriented** avviene a tutti i livelli della fase di sviluppo oggi tutti i team di sviluppo adottano come strumento per realizzarla il linguaggio **UML** che, come di seguito descritto, oltre a essere un linguaggio di **modellazione** è anche un linguaggio di **progetto**.

Il linguaggio **UML** fornisce delle notazioni adatte a esprimere dei concetti propri della descrizione architettonale, come la struttura fisica dello hardware e del software.

## ■ Le schede CRC per le classi e le associazioni tra di esse

Ricordiamo che la classe nella OOP rappresenta un “**concetto**” dell’applicazione che viene modellata e in essa “include” tutti gli oggetti che hanno lo stesso comportamento, che possono essere:

- un’entità fisica, come ad esempio un’automobile, un calcolatore ecc.;
- un’entità del settore applicativo: fattura, conto corrente ecc.;
- un’entità logica: un gestore di ordini;
- un’entità applicativa: un pulsante sull’interfaccia;
- un’entità “informatica”: una tabella hash, un record.

Quando si utilizza il processo di progettazione orientato agli oggetti una fase cruciale è proprio quella di **identificazione delle classi** e di descrizione del **comportamento di ciascuna classe** e delle **relazioni esistenti** tra di esse.

I metodi più usati per individuare le **classi** sono due:

- analisi **nome-verbo**;
- analisi **CRC**.

### Analisi nome-verbo

L’individuazione della classi mediante l’analisi **nome-verbo** avviene analizzando tutta la documentazione disponibile prodotta in fase di pre-analisi e analisi e selezionando su di essa i nomi e verbi:

- i **nomi** e i “**predicati nominali**” sono i potenziali candidati per divenire **classi** o **attributi**;
- i **verbi** sono potenziali candidati a divenire **responsabilità** di classe.

La trattazione di questa tecnica viene effettuata nei corsi di programmazione.

### Schede CRC

Un ottimo metodo per documentare i legami tra le **classi** è quello di utilizzare le **schede CRC**, dove **CRC** sta per **classi**, **responsabilità**, **collaboratori** in quanto in queste schede vengono messi in evidenza questi elementi.

Proposte da **Kent Beck** e **Ward Cunningham** nel 1989 come strumento didattico per insegnare la progettazione Object-Oriented utilizzano un **metodo di brainstorming** iterativo che coinvolge sviluppatori, esperti e committenti.

Si deve produrre una scheda **CRC** per ogni classe che è stata individuata: graficamente la scheda può avere un formato simile a quello riportato in figura (pagina seguente) e generalmente viene realizzata in formato cartaceo, in modo da avere “sotto mano” tutte le schede per poterle consultare e aggiornare contemporaneamente.

Le schede sono realizzate appositamente di piccole dimensioni (generalmente in cartoncino di formato 4" x 6") così da “obbligare” i progettisti a limitare la complessità della descrizione e a inserire solo le nozioni indispensabili senza aggiungere troppe informazioni di dettaglio; anche le righe sono poche, in modo da non dare alle singole classi troppe responsabilità.

L’elenco delle responsabilità della classe non sono altro che le azioni che la classe deve effettuare e i **collaboratori** sono i “nominativi” delle altre classi che sono coinvolte in quella azione.

Un esempio del formato di una scheda **CRC** è il seguente:

Nome della classe	Superclasse	Sottoclassi
Descrizione responsabilità	Collaboratori	

Durante le fasi di **brainstorming**, grazie al loro formato ridotto, le **CRC** consentono di effettuare una serie di attività gestuali utili, come piazzare le carte su un tavolo e spostarle, riorganizzarle, o anche eliminarle e sostituirle facilmente con altre nel corso della discussione.

Anche le modalità di posizionamento delle carte stesse sul tavolo può essere usato per avere visivamente informazioni aggiuntive sulle relazioni tra le classi: ad esempio, la sovrapposizione di due carte può indicare una loro **collaborazione** mentre il loro posizionamento distante indica la **mancanza di interazione**.

Le carte vengono anche utilizzate nelle fasi successive della progettazione, come per esempio nelle verifiche dei **casi d'uso** e degli **scenari**, che saranno descritti nel secondo volume di questo corso.

### ESEMPIO

Vediamo ad esempio una scheda **CRC** per una situazione tassista-cliente.

La situazione è semplice: il cliente sale sul taxi e indica all'autista la destinazione, il tassista avvia il tassametro, inizia a guidare fino al raggiungimento della meta, ferma il tassametro e porge il conto al cliente. Questo paga il corrispettivo e scende dall'auto.

Le classi individuate sono tre: tassista, cliente e tassametro e quindi realizzeremo tre schede **CRC**.

Per la loro compilazione, come prima cosa si scrivono i verbi sulla scheda della **classe** che compie questa azione e quindi ne ha la **responsabilità**: a fianco si indicano i soggetti che sono “coinvolti” in tale azione, cioè chi ne chiede e/o ottiene i risultati oppure è necessaria sono necessarie per portare a termine quella particolare operazione (**collaboratori**).

Le tre schede del nostro semplice esempio sono le seguenti:

Nome della classe	Superclasse	Sottoclassi
Cliente	Persona	
Descrizione responsabilità	Collaboratori	
Chiama il taxi	tassista	
Sale sul taxi	tassista	
Indica la destinazione	tassista	
Paga il corrispettivo	tassista, tassametro	
Scende dal taxi	tassista	

Nome della classe	Superclasse	Sottoclassi
Tassista	Automobilista	
<b>Descrizione responsabilità</b>	<b>Collaboratori</b>	
Riceve il cliente	cliente	
Acquisisce la destinazione	cliente	
Guida il taxi a destinazione	cliente	
Avvia il tassametro	tassametro	
Ferma il tassametro	tassametro	
Chiede il corrispettivo	cliente, tassametro	

Nome della classe	Superclasse	Sottoclassi
Tassametro	Orologio	
<b>Descrizione responsabilità</b>	<b>Collaboratori</b>	
Indica il prezzo	tassista, cliente	
Viene acceso e spento	tassista	

Una volta realizzate le schede, queste vengono collocate sul tavolo e si controllano assieme agli altri membri del gruppo di lavoro per verificare se si “è sulla strada giusta”: si prende una scheda alla volta e si simulano le attività, verificando che tutte le classi coinvolte siano state correttamente indicate tra i collaboratori.



### Prova adesso!

- Compilazione schede CRC

Compilare e discutere le schede **CRC** per rappresentare la seguente situazione.

Una gelateria è frequentata dai clienti e dal personale: il personale raccoglie gli ordini, prepara e consegna i gelati (cono o coppette). Il pagamento avviene alla cassa e al cliente viene consegnato un apposito scontrino con l'elenco dei beni acquistati. Un membro del personale si occupa del controllo sulle giacenze dei gelati e quando un gusto termina, aggiorna l'elenco dei gusti disponibili e invia un ordine in cucina per la produzione del gusto esaurito.

## ■ Il linguaggio di modellazione UML

Tra i vari problemi che ci sono nell'affrontare un progetto software dobbiamo sottolineare quelli legati alla comunicazione; infatti nelle diverse fasi di sviluppo è necessario un assiduo dialogo tra i progettisti, tra i progettisti e il committente, tra i progettisti presenti e i progettisti futuri.

Non è possibile utilizzare il **linguaggio naturale** in quanto questo è troppo impreciso e si presta a interpretazioni soggettive; d'altra parte neppure il **codice sorgente** può essere utilizzato anche se è preciso, ma troppo dettagliato e non comprensibile a tutti i soggetti coinvolti.

La soluzione migliore è utilizzare un **linguaggio di modellazione** che unisca i pregi e “attenui” i difetti dei linguaggi prima indicati, cioè sia sufficientemente preciso e flessibile dal punto di vista descrittivo per poter arrivare a un qualunque livello di dettaglio e possibilmente standard, così che possa essere compreso anche se avvenisse la sostituzione di qualche membro del gruppo di sviluppo.

Tra i **linguaggi di modellazione** il più utilizzato è l'**UML** (*Unified Modeling Language*), che è un linguaggio potente, completo, semplice e naturale, ormai universalmente riconosciuto come uno standard de facto a livello mondiale.

L'**UML** consiste in un insieme di notazioni sviluppate a partire dagli anni Ottanta e consolidate in un apparato standard alla fine degli anni Novanta 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**.

Tra le caratteristiche principali dell'**UML** possiamo ricordare che:

- è indipendente dal processo, anche se promuove uno stile di analisi e progettazione:
  - guidato dai casi d'uso (**use case driven**), cioè dalle funzionalità che il sistema deve fornire;
  - **iterativo e incrementale**, quindi guidato dall'analisi dei rischi (**risk-driven**);
- prevede meccanismi di estensione del linguaggio stesso:
  - è integrabile con altre tecniche;
  - è orientato agli oggetti;
- è uno standard **Object Management Group (OMG)**, consorzio creato nel 1989 con 440 aziende con l'obiettivo di creare un sistema di gestione di un'architettura distribuita

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 che 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.

Dato che è necessario esplicitare le **relazioni** esistenti tra le **classi** stesse e la struttura gerarchica che viene realizzata attraverso la realizzazione di sottoclassi nel diagramma delle **classi** sono disponibili gli strumenti grafici per mettere in relazione tutte le classi.

L'**UML** è quindi un linguaggio di modellazione grafico che permette di creare varie descrizioni più o meno astratte di un sistema.

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 della 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 e 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 dal 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.

## 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:**

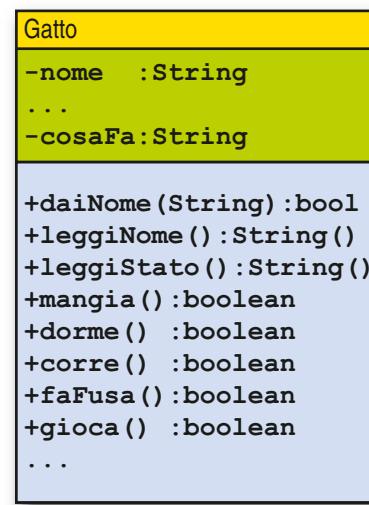
- in ogni caso con le iniziali in minuscolo (per esempio, **veloce()** e **mangiaPizza()**).

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:

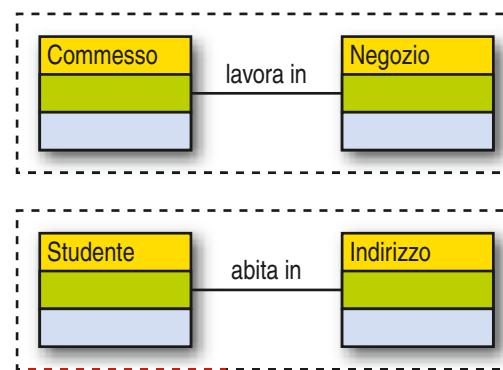
- **semplice**;
- **complessa**;
- **riflessiva**;
- **forte**;
- **debole**.

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

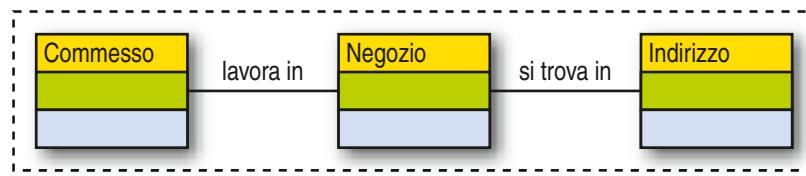
Il concetto di relazione è quello classico della matematica (in particolare della teoria degli insiemi), derivante dal termine inglese **relationship**, anche tradotto con **associazione** o **correlazione**.

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
	<b>Relazione</b> Le relazioni vengono rappresentate con una linea che congiunge le classi
	<b>Opzionalità</b> 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
	<b>Obbligatorietà</b> È rappresentata da una linea continua e indica che tutte le istanze di C1 devono essere associate ad almeno una istanza di C2
	<b>Cardinalità</b> Relativamente a quanto contenuto nel cerchio tratteggiato, per una o più istanze di C1, l'associazione individua una sola istanza di C2
	<b>Cardinalità</b> Relativamente a quanto contenuto nel cerchio tratteggiato, per una o più istanze di C1, l'associazione individua una o più istanze di C2

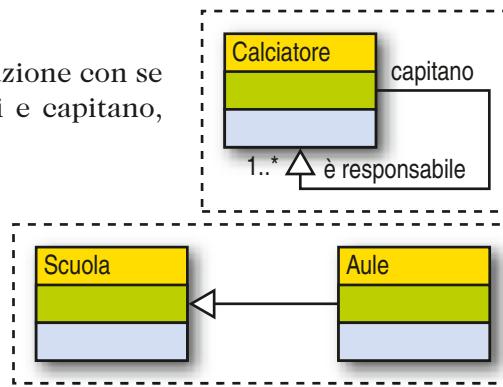
## Relazione riflessiva

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)**.

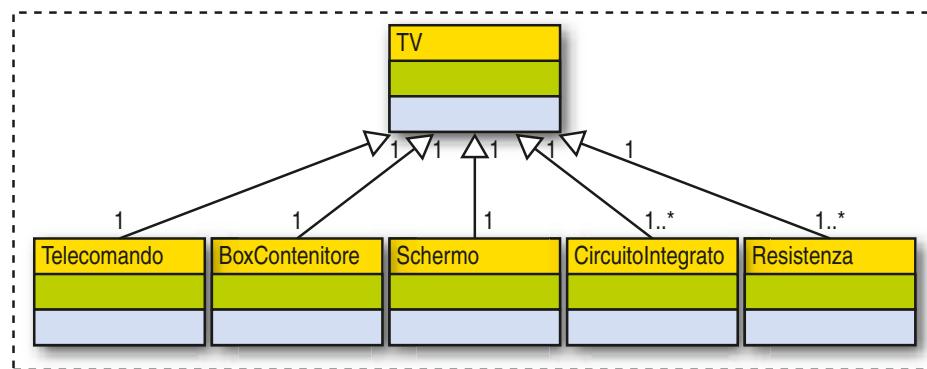
Una linea unisce “l’intero” per un componente con un triangolo raffigurato sulla linea stessa vicino all’“intero”: nell’esempio della figura l’intero è la scuola che è composta di aule. Nel caso di una aggregazione con diversi componenti è possibile effettuare una rappresentazione gerarchica in cui l’“intero” si trova in cima e i componenti (“parte”) al di sotto.



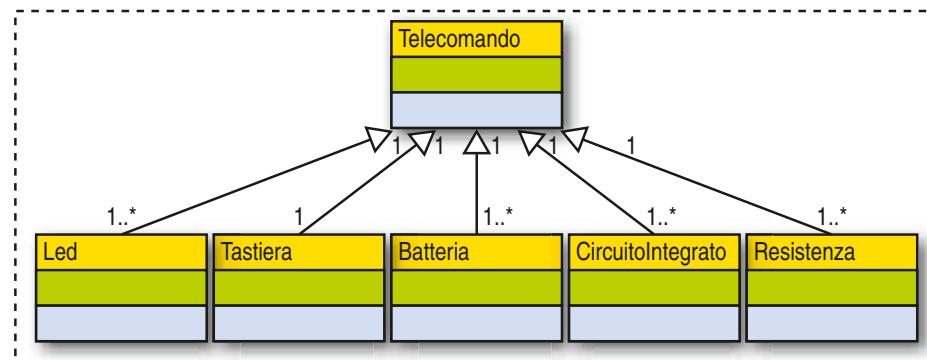
### ESEMPIO

Vediamo un esempio completo e articolato di aggregazione: un televisore.

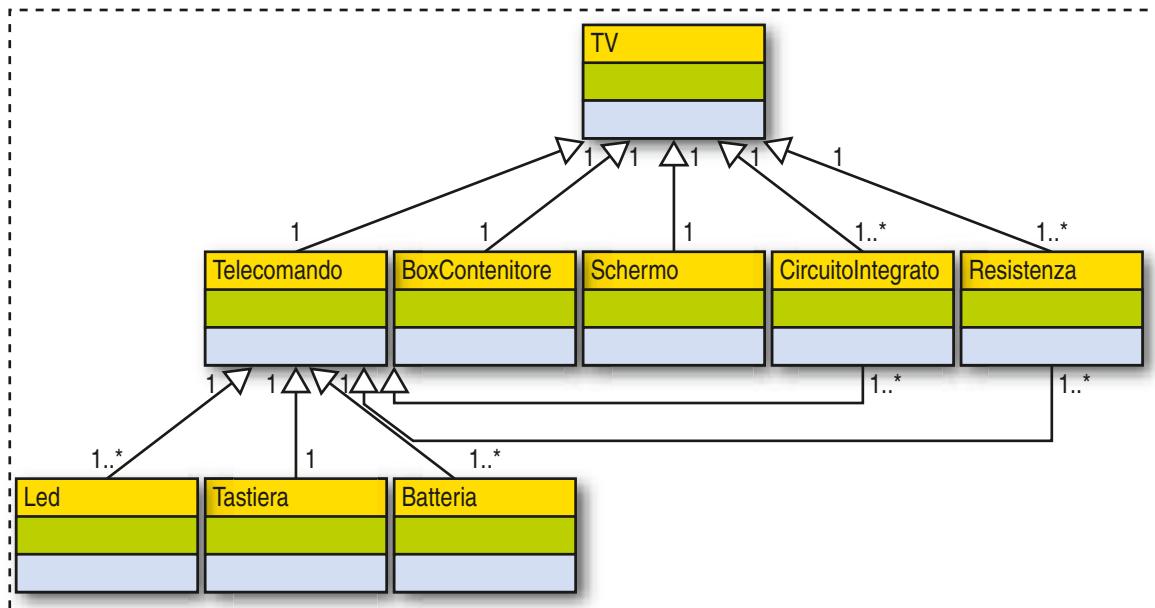
Innanzitutto individuiamo le “parti” che costituiscono un televisore (senza entrare nei dettagli): ogni TV ha un involucro esterno, uno schermo, degli altoparlanti, delle resistenze, dei transistor, alcuni circuiti integrati, un telecomando e altri componenti che trascuteremo in questo esempio. Il suo 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:



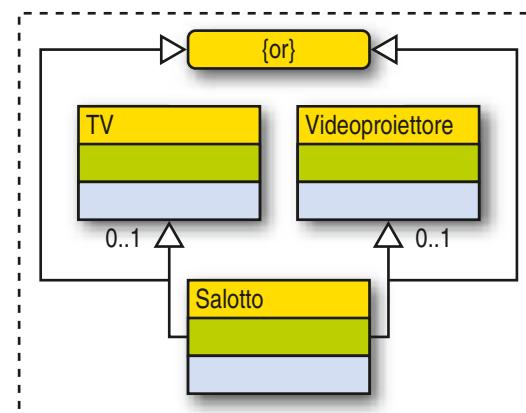
Qualche volta l'insieme di possibili componenti di un'aggregazione si identifica in una relazione OR (oppure OR esclusivo XOR): per modellare tale situazione è necessario utilizzare una **constraint**, che consiste nella parola OR all'interno di parentesi graffe su una linea tratteggiata che connette 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. ►



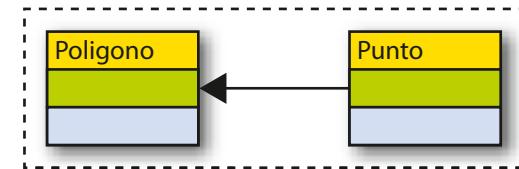
### AGGREGAZIONE

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)

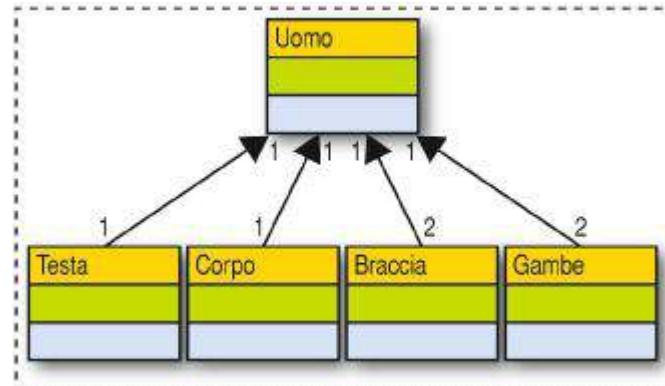
Una **composizione** è un tipo più forte di aggregazione in quanto ogni componente può appartenere soltanto a un “intero”: nel caso per esempio di un Poligono, un Punto non può appartenere contemporaneamente a due Poligoni.



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.

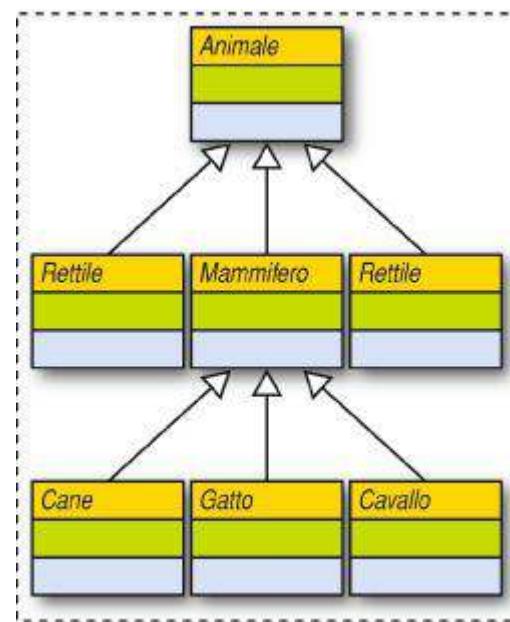


## Ereditarietà e generalizzazione

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

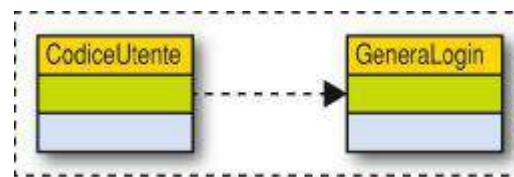
Un ultimo aspetto di relazione tra le classi che necessita di essere riportato è la dipendenza, che avviene tra due classi quando la modifica di una classe crea (o potrebbe creare) qualche conseguenza sull'altra. La classe "autonoma" prende il nome di **fornitore** (o **supplier**) e la classe dipendente 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**

Realizzare il diagramma UML delle classi Veicolo, Automobile e Pneumatico.

**Soluzione**

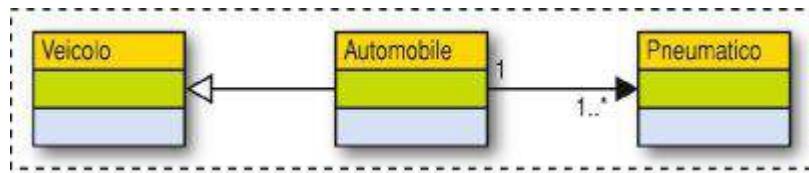
Scriviamo “a parole” il legame esistente tra le classi:

“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.

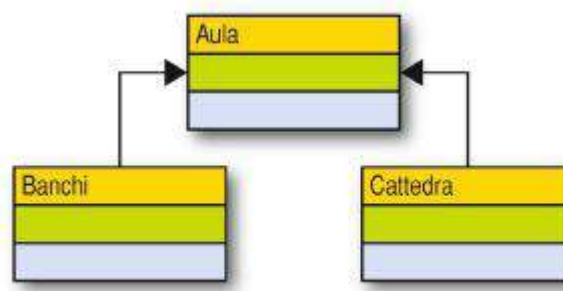
“Un’aula è un **locale** che **ha un nome** (IV°I3, laboratorio1 ecc.) ed è **composta** da una **cattedra** e  $n$  **banchi**; può **ospitare**  $m$  **corsi** (informatica, sistemi, inglese ecc.) e a ogni corso è **assegnato** uno e un solo **docente** ed è **seguito** da uno o più **studenti**.

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:

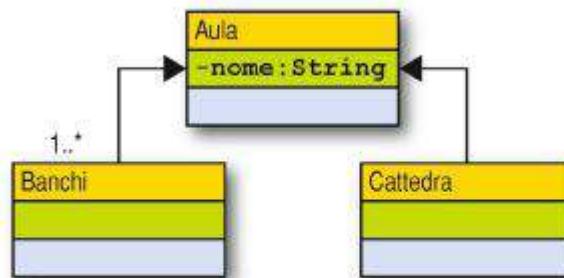
“Un’aula è un **locale** che **ha un nome** ed è **composta** da  $n$  **banchi** e una **cattedra**” si ottengono tre classi (Aula, Banchi, Cattedra) che hanno tra loro relazione di associazione forte (composizione) in quanto ogni componente può appartenere soltanto a un “intero”, cioè ogni cattedra 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.



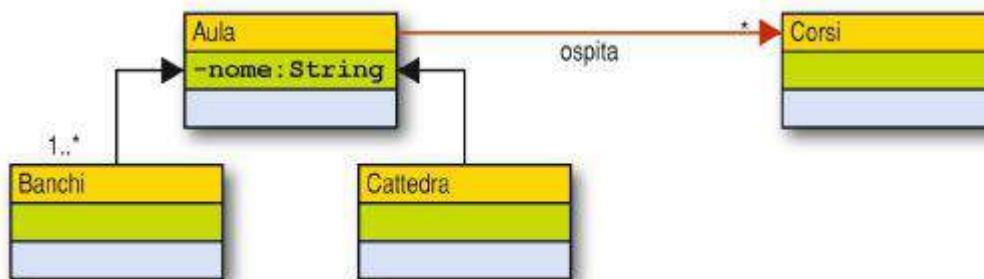
## Passo 3: analisi seconda riga e completamento classi

“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.”

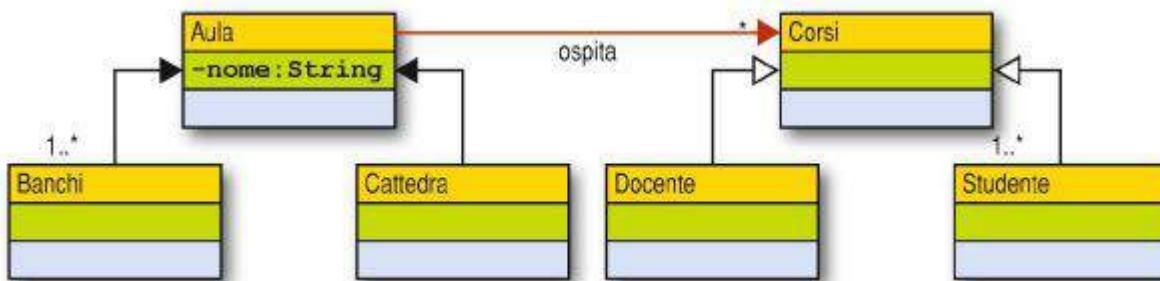
Due nuove classi, la classe **Docenti** e **Alunni**, con molteplicità rispettivamente 1 a 1 e 1 o molti con la classe **Corsi**.



## Passo 4: analisi ultima riga

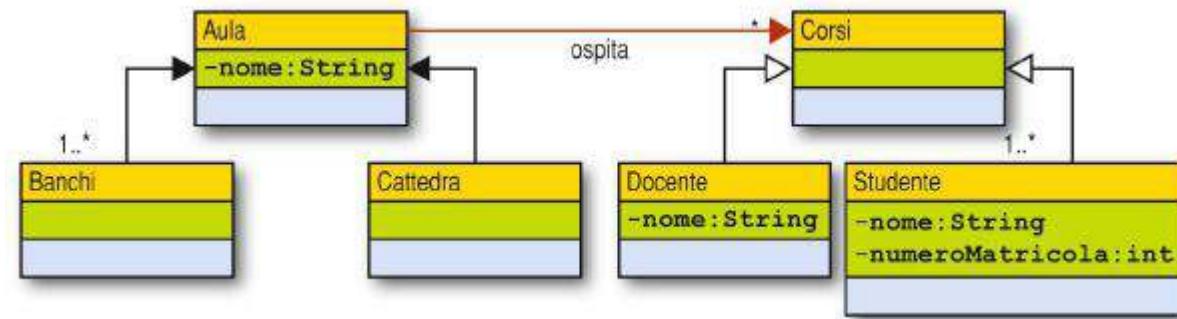
“Il **docente** e gli **studenti** hanno un **nome** e gli **studenti** hanno anche una **matricola**.”

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 **Studente**; 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.



## ■ Conclusioni

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.

### AREA digitale

La qualità del software con la norma ISO 9000



#### 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

### 1. Risposta multipla

**1 L'assistenza può essere:**

- a. teorica/tecnica
- b. teorica/operativa
- c. operativa/tecnica
- d. pratica/teorica

**2 La documentazione riservata all'utente consiste in (indicare la risposta errata):**

- a. documentazione inerente all'installazione del prodotto
- b. documentazione inerente all'avvio dell'utilizzo del prodotto
- c. documentazione inerente alla struttura degli archivi
- d. documentazione interna
- e. manuale d'uso

**3 La documentazione tecnica consiste in:**

- a. documentazione interna/esterna
- b. documentazione on line/esterna
- c. documentazione pratica/teorica
- d. documentazione utente/programmatore

**4 L'acronimo UML significa:**

- a. Uniform Module Language
- b. Unified Module Language
- c. Uniform Modeling Language
- d. Unified Modeling Language

**5 I diagrammi di base dell'UML sono:**

- a. 6
- b. 7
- c. 8
- d. 9

**6 Quale tra i seguenti non è un diagramma di base dell'UML?**

- a. Class Diagram
- b. Object Diagram
- c. Method Diagram
- d. State Diagram
- e. Activity Diagram
- f. Component Diagram
- g. Deployment Diagram

**7 Le associazioni possono essere (indicare quella errata):**

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>a. deboli</li> <li>b. forti</li> <li>c. complesse</li> </ul> | <ul style="list-style-type: none"> <li>d. semplici</li> <li>e. oggettive</li> <li>f. riflessive</li> </ul> |
|---|--|

**8 Quale tra le seguenti non è una forma di cardinalità della OOP?**

- a. zero a zero
- b. uno a zero
- c. uno a uno
- d. uno a molti

### 2. Vero o falso

**1 La documentazione inerente all'installazione fa parte della documentazione utente.**

**V F**

**2 La documentazione esterna all'installazione 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

### 1. Esercizi

Compilare e discutere le schede **CRC** per rappresentare le seguenti situazioni.

- 1 Una festa domestica viene organizzata dai padroni di casa invitando degli amici (ospiti) che portano un presente alla padrona di casa. Gli inviti vengono spediti per e-mail oppure per telefono e contengono data e ora della festa ed eventualmente la ricorrenza festeggiata o il tema della serata.  
Durante la festa vengono somministrate bevande e alimenti, sempre con un sottofondo musicale adatto a ogni singola fase della serata.
- 2 Un Comune vuole realizzare uno sportello automatico da cui i cittadini possono ritirare certificati o pagare multe, previa autenticazione tramite tessera magnetica o inserimento di un PIN personale.
- 3 Si vuole realizzare un programma per le operazioni tra polinomi dove un polinomio è dato dalla somma di uno o più monomi e un monomio è caratterizzato da un segno, un coefficiente e un grado.  
Il programma deve leggere il nome dei due polinomi ed effettuare le operazioni richieste.
- 4 Un ambulatorio veterinario vuole gestire le prenotazioni dei propri pazienti: ogni animale ha uno (o più) padroni che devono autenticarsi per accedere al servizio; oltre alla prenotazione sono disponibili anche le operazioni relative alla modifica dei dati anagrafici del cliente e del paziente, alla visualizzazione dei suoi dati anagrafici, alla cancellazione della scheda anagrafica.
- 5 Una rivendita di auto usate vuole catalogare ogni automezzo con una scheda che riporta l'anno di immatricolazione, il numero di chilometri e la data dell'ultima revisione. Ogni automezzo viene classificato in diverse categorie in base alla tipologia e alla possibilità o meno di trasportare persone.  
Devono essere memorizzati sia gli acquirenti che i venditori, e per ogni operazione deve essere stipulato un contratto per acquistare uno o più automezzi.

### 2. Esercizi

Genera un diagramma UML dell'implementazione delle classi.

- 1 Descrivi il diagramma delle classi di un impianto Hi-Fi, con livello di dettaglio fino ai componenti elettronici.
- 2 Descrivi il diagramma delle classi di un'automobile, ereditando dalla classe **Automezzi**, dalla classe Cerchio e dalla classe **Sedia**.
- 3 Descrivi il diagramma delle classi di un complesso musicale (comprensivo di musicisti), ereditando dalla classe **Strumenti**, dalla classe **ApparecchiatureElettroniche** e dalla classe **Umano**.
- 4 Descrivi il diagramma delle classi di un circo, comprendendo animali e artisti, ereditando proprio dalle classi **Animali** e **Artisti**.
- 5 Descrivi il diagramma delle classi di un videogioco tipo Battaglia Navale ereditando dalle classi **MezzoDiTrasporto** e **Arma**.
- 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.
- 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.
- 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.



# ESERCITAZIONI DI LABORATORIO 1

## I DIAGRAMMI UML CON ARGOUML

### ■ Strumenti di la realizzazione di diagrammi UML

L'**UML** (*Unified Modeling Language*) è un linguaggio di modellazione unificato creato principalmente per la progettazione di software a oggetti ma viene anche utilizzato per la realizzazione dei diagrammi **E-R** nella fase di progettazione concettuale dei database.

UML è stato progettato da **OMG** (*Object Management Group*, sito di riferimento <http://www.omg.org/>), un consorzio che ne ha definito le specifiche con l'obiettivo di realizzare un linguaggio che potesse rappresentare un'applicazione (e non necessariamente un software) come un modello, indipendentemente dai dettagli della sua implementazione.

Come ogni linguaggio anche l'**UML** (sito di riferimento <http://www.uml.org/>) necessita di tools appropriati che ne agevolino l'utilizzo e aiutino lo sviluppatore a realizzare i propri diagrammi.

Sono presenti sul mercato tanti tools che comprendono anche la realizzazione di diagrammi, sia gratuiti che a pagamento.

Lo strumento più diffuso a pagamento è **Rational Rose**, disegnato per fornire ai team di sviluppo tutti gli strumenti necessari per il modellamento di soluzioni robuste ed efficienti (sito web alla pagina <http://www.rational.com>).

Anche la Microsoft ha prodotto uno strumento che permette di definire un sottoinsieme dei modelli che il **Rational Rose** mette a disposizione: il **Microsoft Visual Modeler**.

Tra i prodotti software open source ne segnaliamo due, **ArgoUML**, che descriveremo in questa lezione, e **UMLet**: sono entrambi basati su **Java**, permettono la scrittura di diagrammi **UML** e inoltre, tra le opzioni, offrono anche la possibilità di esportare i diagrammi in diversi formati, come **SVG**, **JPG** e **PDF**.

### ■ ArgoUML

**ArgoUML** è un ottimo prodotto software open source che permette di creare diagrammi **UML** che si candida come una valida alternativa ai prodotti commerciali.

**ArgoUML** è un software open source multi-piattaforma sviluppato per fornire un completo editor di diagrammi **UML** dotato di numerose funzionalità.

Tra le principali caratteristiche di **ArgoUML** troviamo il supporto per **XMI** con la possibilità di esportare i grafici nei più diffusi formati come il vettoriale **SVG** oppure come immagini **GIF**, **PNG**, **PS**, **EPS**, e **PGML**.

Il toolkit **Dresden OCL** incluso in **ArgoUML**, consente di “esportare” i nostri progetti in **Java**, **C++**, **C#**, **PHP**, e di supportare altri linguaggi di programmazione.

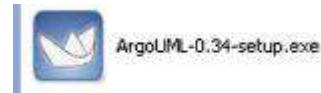
Utile è anche la funzione **Lista di controllo** (o **To Do List**) che ci consente di effettuare un elenco delle cose da fare per il nostro progetto.

**ArgoUML** è un software scritto in Java che possiamo utilizzare in **Linux**, **Windows** e **Mac** senza alcuna installazione: è sufficiente avere installato la **JVM**.

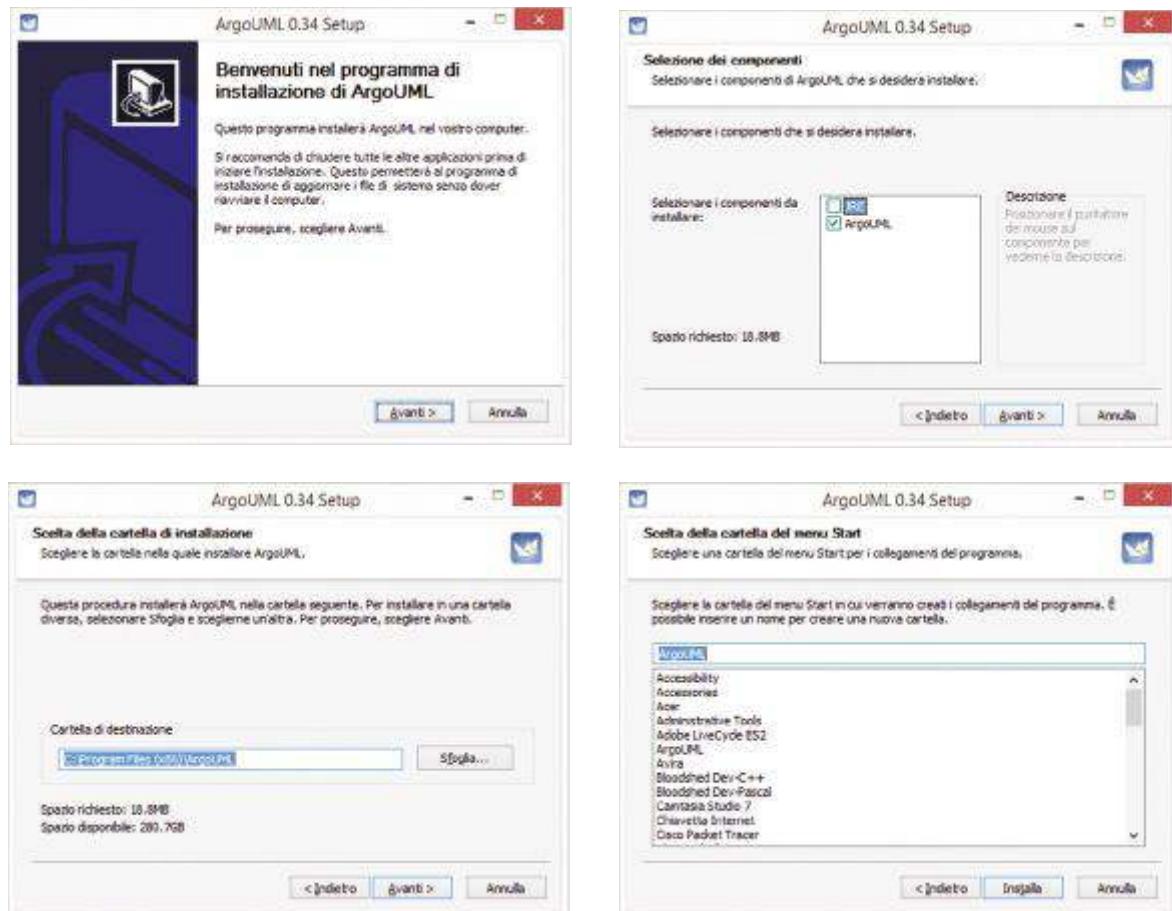
**ArgoUML** può essere scaricato dal sito ufficiale <http://argouml.tigris.org> oppure dalla cartella materiali della sezione del sito [www.hoepliscuola.it](http://www.hoepliscuola.it) dedicata a questo volume.

## Installare ArgoUML

L'installazione avviene in modo automatico; basta cliccare sull'icona corrispondente: ►



Confermiamo le successive videate seguenti che non necessitano di chiarimenti:

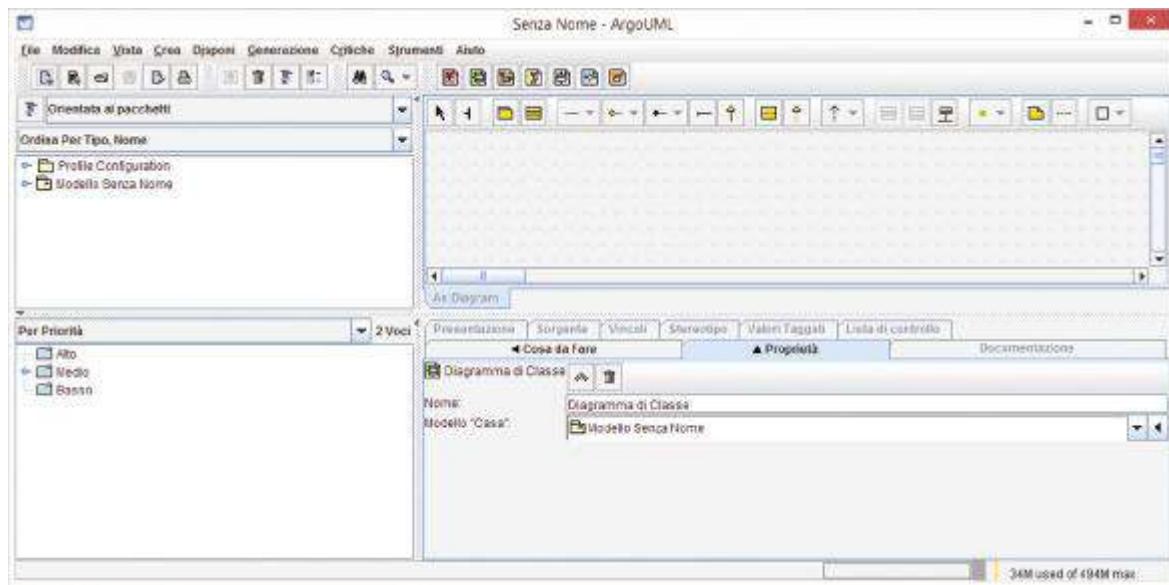


Al completamento dell'installazione clicchiamo su [Fine] e avviamo il programma.



## Avviare ArgoUML

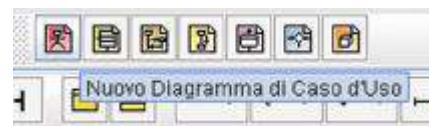
Apriamo **ArgoUML** e cominciamo a vedere assieme come è costituito un diagramma **UML**.



La schermata principale di **ArgoUML** è costituita da un'unica finestra contenente 4 pannelli:

- ▶ i due pannelli di sinistra sono quelli che ci permettono di realizzare i diagrammi;
- ▶ il pannello superiore costituisce l'area di disegno/progetto;
- ▶ il pannello inferiore contiene le tabelle per la definizione delle caratteristiche ed è “sensibile al contesto”, cioè si abilitano automaticamente solo le opzioni disponibili per la fase di editing che stiamo effettuando.

Possiamo selezionare il diagramma che ci interessa realizzare tramite le icone della barra principale in modo da avere nei menu le opzioni specifiche per quel diagramma:



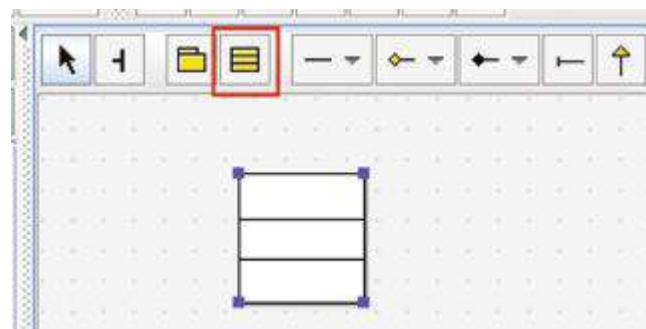
Nella **Tool Bar** presente sopra il pannello con la griglia sono disponibili tutti i tipi di elementi che possiamo aggiungere nel diagramma: basta passare sopra col puntatore del mouse e i Tool Tips che appaiono ci indicano la loro funzionalità.

Le icone di alcuni strumenti sono affiancati da una freccetta nera rivolta verso il basso, se la cliccate appariranno al di sotto altri tipi dello stesso strumento.



## Creiamo una classe

Per creare una classe clicchiamo sull'icona della **Tool-Bar** per selezionarla e quindi nel mezzo della pagina del diagramma, ottenendo quanto segue:



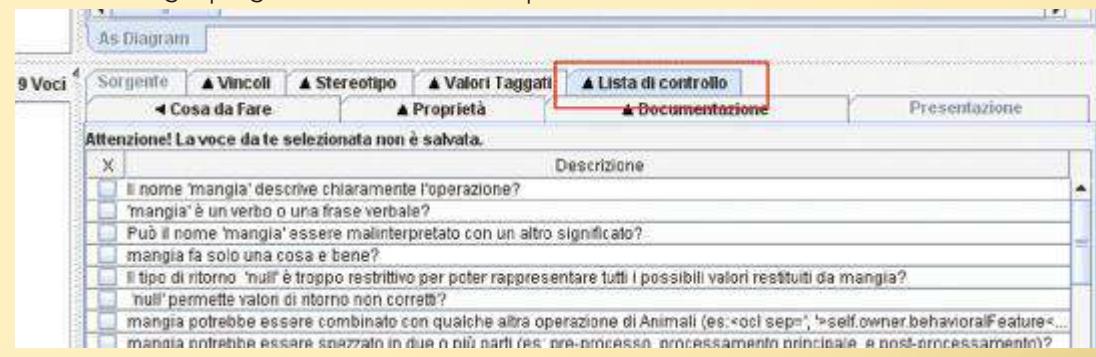
Procediamo assegnando un nome nella tabella proprietà della finestra inferiore (nel nostro caso **Animali**):



e aggiungendo alcuni attributi e proprietà selezionando l'icona corrispondente: otterremo qualcosa simile al seguente diagramma:



**ArgoUML** non è solo uno strumento per disegnare diagrammi, ma è molto utile in fase di progettazione: provate a selezionare l'icona [Lista di controllo] e otterrete un insieme di domande che ogni programmatore dovrebbe porsi durante la realizzazione del software.



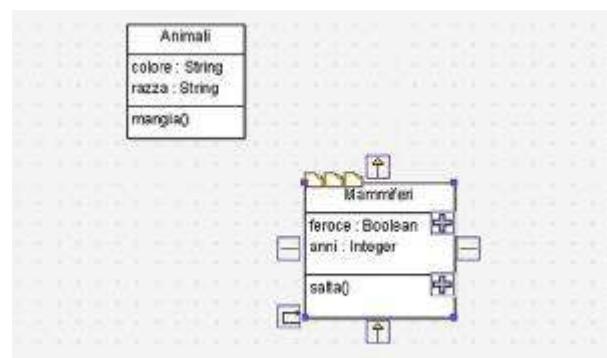
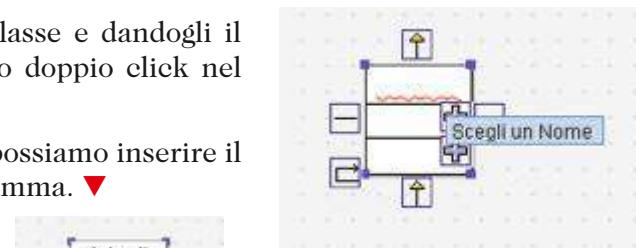
## Le associazioni

Dopo aver salvato la prima classe, creiamo una seconda classe in modo da poter vedere come si realizzano le associazioni; ci sono diversi tipi di associazione in UML: l'**associazione semplice**, l'**aggregazione**, la **composizione** e la **generalizzazione**.

Il tipo di associazione più classico è la **generalizzazione**, che rappresenta il concetto di **ereditarietà** della **OOP**: si disegna con la freccia che parte dalla classe figlia e va verso quella del "parente".

Iniziamo, quindi, creando una nuova classe e dandogli il nome direttamente sul disegno, facendo doppio click nel primo riquadro della classe. ►

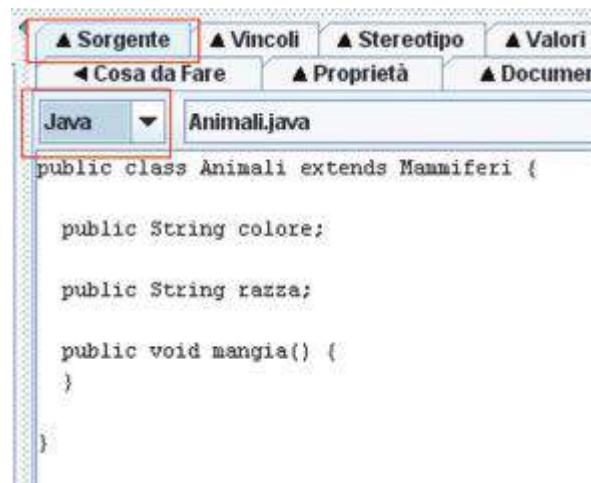
Comparirà un campo di testo nel quale possiamo inserire il nome direttamente all'interno del diagramma. ▼



Colleghiamo ora le due classi tramite un'associazione di tipo **generalizzazione**: selezioniamo la classe **Mammiferi** e attendiamo qualche istante lasciando il puntatore del mouse su di essa. Vengono visualizzate delle "zone sensibili" come si può vedere nella figura a sinistra.

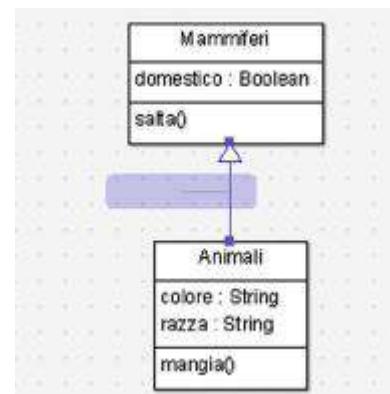
Per creare la **generalizzazione** dovremo trascinare la frecia gialla, comparsa in alto al centro di **Mammiferi**, fino ad **Animali**, ottenendo il risultato mostrato a fianco: ▶

Selezioniamo ora **Animali** e clicchiamo la pagina **Sorgente** dentro al pannello inferiore: **ArgoUML** genera anche il codice sorgente nel linguaggio selezionato nella tendina del menu dove è possibile scegliere tra **Java**, **PHP5**, **SQL**, **CSharp**, **C++**. ▼

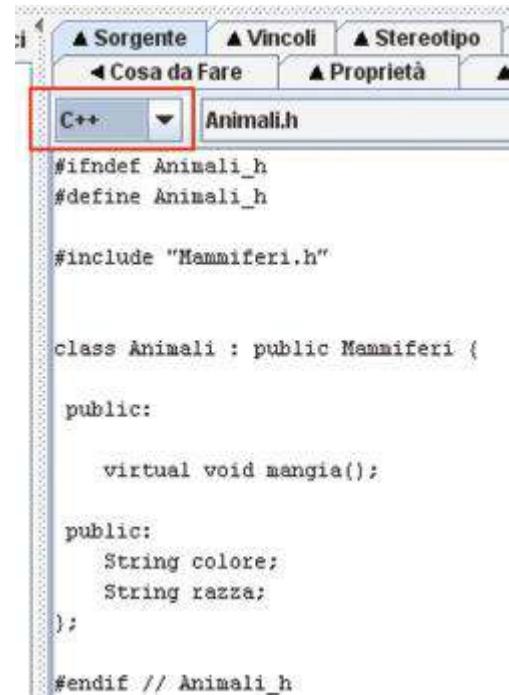


The screenshot shows the ArgoUML interface with the UML class diagram on the right. The diagram consists of two classes: **Mammiferi** (superclass) and **Animali** (subclass). **Mammiferi** has a Boolean attribute **domestico** and a method **saita()**. **Animali** has a String attribute **colore**, a String attribute **razza**, and a method **mangia()**. A generalization arrow points from **Mammiferi** to **Animali**. On the left, there is a tool bar with tabs like **Sorgente**, **Vincoli**, **Stereotipo**, **Valori 1**, **Cosa da Fare**, **Proprietà**, and **Documenti**. The **Sorgente** tab is selected and highlighted with a red box. Below it, a dropdown menu shows **Java** as the selected language, also highlighted with a red box. The generated Java code is displayed in the main pane:

```
public class Animali extends Mammiferi {
    public String colore;
    public String razza;
    public void mangia() {
    }
}
```



Selezioniamo ora dall'elenco un nuovo linguaggio di programmazione, ad esempio **C++**, e otteniamo la seguente codifica: ▼



The screenshot shows the ArgoUML interface with the UML class diagram on the right. The code editor on the left displays the generated C++ header code for **Animali.h**. The code includes the preprocessor directive **#ifndef Animali\_h**, the definition **#define Animali\_h**, the inclusion of **"Mammiferi.h"**, the class definition **class Animali : public Mammiferi {**, the public section with the virtual method **virtual void mangia();**, another public section with attributes **String colore;** and **String razza;**, and the closing brace **};**. The **Sorgente** tab is selected and highlighted with a red box. Below it, a dropdown menu shows **C++** as the selected language, also highlighted with a red box.

```
#ifndef Animali_h
#define Animali_h

#include "Mammiferi.h"

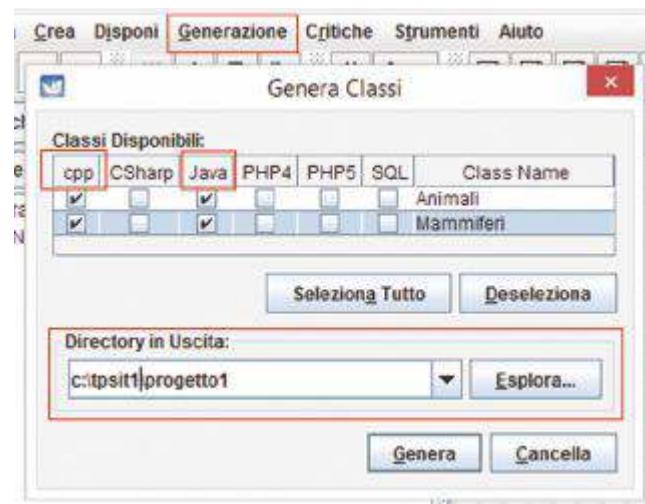
class Animali : public Mammiferi {

public:
    virtual void mangia();

public:
    String colore;
    String razza;
};

#endif // Animali_h
```

Per generare il codice basta selezionare dal menu generale l'opzione **Generazione** e scegliere la classe desiderata e il percorso di memorizzazione: ▼

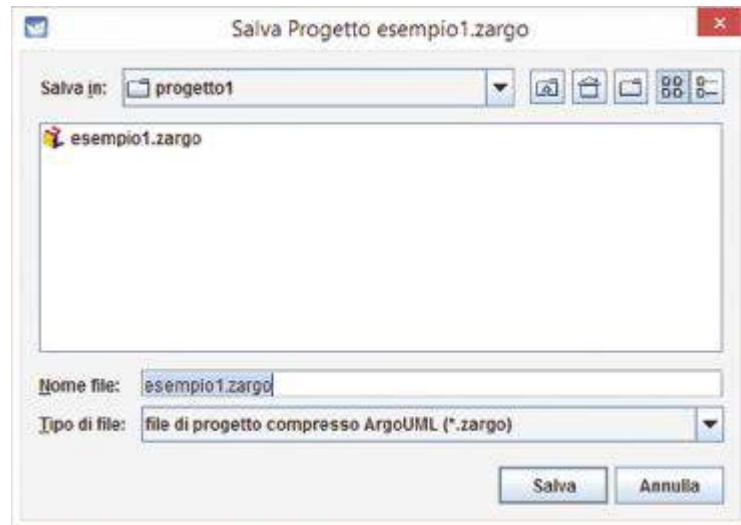


Nel nostro esempio otteniamo:

Nome	Ultima modifica	Tipo	Dimensione	
esempio1.zargo	18/11/2014 18:24	ArgoUML Project ...	6 KB	
Animali.h	18/11/2014 18:29	C Header File	1 KB	
Mammiferi.h	18/11/2014 18:29	C Header File	1 KB	
Animali.cpp	18/11/2014 18:29	C++ Source File	1 KB	
Mammiferi.cpp	18/11/2014 18:29	C++ Source File	1 KB	
Animali.java	18/11/2014 18:29	File JAVA	1 KB	
Mammiferi.java	18/11/2014 18:29	File JAVA	1 KB	

## ■ Conclusioni

Il progetto viene semplicemente salvato in formato **UML** assegnandogli un nome: il suffisso che viene aggiunto automaticamente è **.zargo**.



È comunque preferibile effettuare il salvataggio in formato **XMI** che è più “stabile”, selezionandolo dalla prima tendina del menu:



► **UML** ► è un formato simile a **XMI**, in cui però si perdono alcune informazioni del diagramma in merito all’aspetto grafico e alla posizione degli elementi nel diagramma.

► **XMI** (acronimo di **XML Metadata Interchange**) è il formato standardizzato da **OMG** definire, scambiare, manipolare e integrare oggetti elaborando dati in formato **XMI**; generalmente viene utilizzato per scambiare dati relativi a modelli **UML** ma si può usare per descrivere tutti i **metadati** che possono essere espressi in linguaggio **Meta-Object Facility (MOF)** utilizzato per la definizione di **metamodelli**. ►

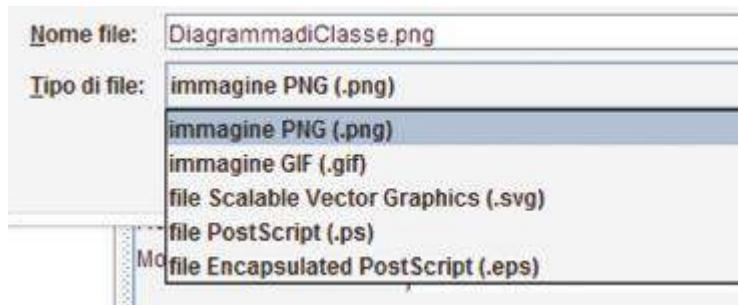
## AREA digitale



L'UML come metamodello

Sempre nello stesso menu è possibile esportare i diagrammi in formato grafico: ►

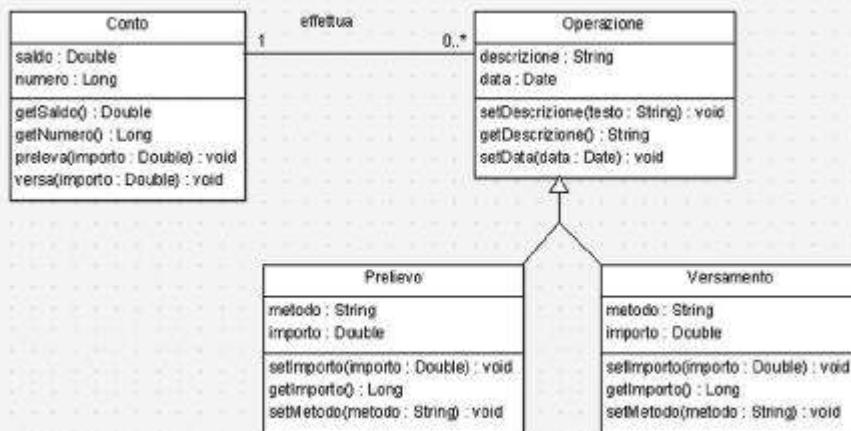
In particolare, il file generato di default è **.PNG**, ma è possibile selezionare i seguenti formati:



### Prova adesso!

- Class diagram
- Uses case

- 1 Crea un nuovo progetto e realizza il seguente diagramma per la **gestione di un conto corrente**.



- 2 Visualizza il relativo codice **Java** e **C++** e salvalo nella cartella **c:\tspit1\progetto2**.
- 3 Quindi realizza i seguenti tre casi d'uso:
  - ▶ visualizza estratto conto;
  - ▶ effettua versamento;
  - ▶ effettua prelievo.

# ESERCITAZIONI DI LABORATORIO 2

## IL MODEL-ORIENTED PROGRAMMING CON UMLE

### ■ Il Model-Oriented Programming

Umples è uno strumento di modellazione e linguaggio di programmazione della famiglia Model-Oriented Programming MOP, creato per aiutare i programmatore nella fase di modellazione.

Umples sta per UML programming language, cioè “linguaggio di programmazione UML”, “ampio” e “semplice”: è una tecnologia per la programmazione orientata al modello (MOP) dove gli sviluppatori del modello “contemporaneamente” realizzano il programma.

Ciò consente agli sviluppatori di lavorare al livello elevato di astrazione consentita dai modelli, sfruttando allo stesso tempo tutti gli strumenti che sono stati sviluppati per chi scrive codice sorgente tradizionale. Umples permette anche di modificare e generare diagrammi modello (macchine a stati e diagrammi delle classi).

Il progetto Umples comprende un editor grafico per disegnare in formato UML le classi e le relazioni tra di esse, un compilatore, un plugin per ◀ Eclipse ▶ e uno strumento a riga di comando.

Partendo dai diagrammi delle classi UML, Umples aggiunge automaticamente le associazioni, gli attributi e creando il codice per i linguaggi di programmazione object-oriented come Java, C++, PHP e Ruby.



◀ Eclipse è un ambiente di sviluppo integrato multilinguaggio e multipiattaforma. Ideato da un consorzio di grandi società. Eclipse può essere utilizzato per la produzione di software di vario genere: si passa infatti da un completo IDE per il linguaggio Java a un ambiente di sviluppo per il linguaggio C++ e a un plug-in che permettono di gestire XML, Javascript, PHP. ▶

Umples è un progetto open source, quindi in via di continua crescita ed evoluzione.

A oggi Umples è pronto per essere usato per la realizzazione di qualsiasi progetto Java, C++ o progetto PHP: il progetto del compilatore Umples stesso è stato scritto in Umples!

Il codice creato è numerato, facilmente leggibile: Umples risparmia al programmatore di dover codificare molte linee di codice “boiler plate” che essendo generate automaticamente sono sicuramente meno soggette a errori.

Il sito di riferimento è all'indirizzo [www.umple.org](http://www.umple.org) che viene reindirizzato su <http://cruise.eecs.uottawa.ca/umple/>. La home page è riportata di seguito:

The screenshot shows the Umple.org home page. On the left, there's a sidebar with "Umple Home Page" and "Key Links" including Umple Online, User Manual, Google Code site with News, Wiki, Blog, Additional Examples, Philosophy and Vision, Tutorials, Best practices, Downloads, Official Releases, and Cutting edge command line jar. The main content area features the "UMPLE" logo, navigation buttons for UmpleOnline, Manual, News+Code, and Download, and a text block about Model-Oriented Programming. Below this is a green box labeled "AREA digitale" containing a download icon and the text "Articolo di Timothy Lethbridge su MOP & Umple".

## ■ Un esempio di utilizzo di Umple

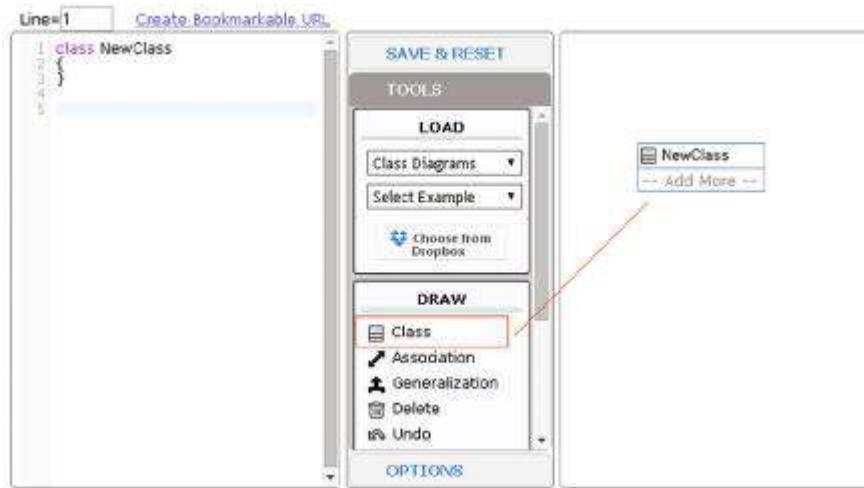
Deseriviamo l'utilizzo di **Umple** mediante un esempio: colleghiamoci all'indirizzo <http://cruise.eecs.uottawa.ca/umpleonline/umple.php> e otteniamo al seguente videata organizzata in tre sezioni:

- nella parte centrale sono presenti i comandi, suddivisi in tre gruppi
  - SAVE & RESET,
  - TOOLS,
  - OPTIONS;
- nella parte di destra realizzeremo graficamente il diagramma UML;
- nella parte di sinistra verrà generato uno pseudo-codice automaticamente.

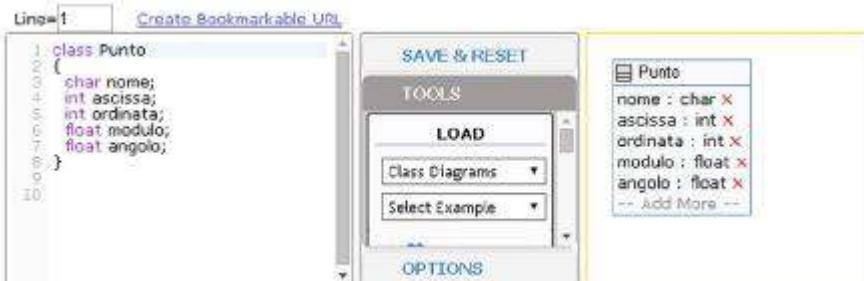
The screenshot shows the Umple Online interface. It has a code editor on the left with placeholder text "Line=1 Create Bookmarkable URL", a "SAVE & RESET" button, and a "TOOLS" palette. The "TOOLS" palette includes sections for "LOAD" (Class Diagrams, Select Example, Choose from Dropbox), "DRAW" (Class, Association, Generalization, Delete), and "OPTIONS". To the right is a large empty area for drawing UML diagrams.

Tra i TOOLS è presente il generatore di codice, che vedremo in seguito: questo farà aprire una finestra inferiore che conterrà il codice scritto nel linguaggio di programmazione selezionato.

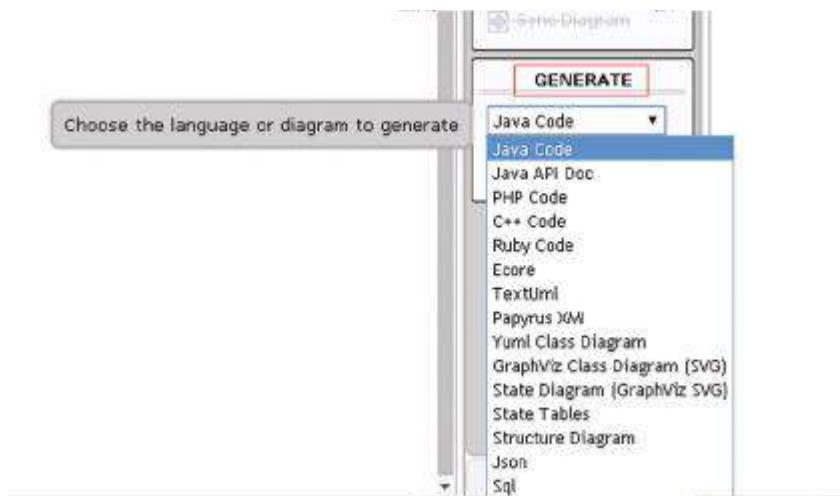
Iniziamo a utilizzare **Ump**le attivando la sezione **TOOLS** e clicchiamo sull'icona **Class** per disegnare una classe: spostando il mouse sull'area di sinistra osserviamo come il puntatore “trascina” un rettangolo **UML**: clicchiamo in un punto a piacere e viene a presentarsi la seguente situazione:



Completiamo la classe: per esempio realizziamo la classe **Punto** (nel piano cartesiano). Dopo aver scritto il nome nella parte superiore aggiungiamo gli attributi sotto riportati osservando che contemporaneamente viene a “formarsi” il codice nell’area di sinistra che “non è ancora il codice in linguaggio di programmazione”, ma un codice in linguaggio **Ump**le.



Per generare il codice in linguaggio di programmazione selezioniamo dalla “tendina di selezione” il linguaggio desiderato, ad esempio Java, e clicchiamo su **[GENERATE]**. ►



Dopo qualche istante la pagina Web ha cambiato aspetto: si è aggiunta una sezione inferiore dove troviamo il codice sorgente in linguaggio Java: ►

Tramite il link indicato è possibile scaricare il codice creato sul proprio computer. Analizziamo succintamente cosa è stato

generato automaticamente: per prima cosa è stata creata la **classe Punto** e vengono definiti i suoi attributi come privati, in linea con le direttive dell'**information hiding**. Quindi viene scritto un **costruttore**:

```

022. //-----
023. // CONSTRUCTOR
024. //-----
025.
026. public Punto(char allome, int aAscissa, int aordinata, float aModulo, float aAngolo)
027. {
028.     nome = allome;
029.     ascissa = aAscissa;
030.     ordinata = aordinata;
031.     modulo = aModulo;
032.     angolo = aAngolo;
033. }
```

Per ogni attributo sono generati i **metodi di interfaccia**, cioè i metodi **getter** e **setter** che permettono l'accesso in lettura e scrittura a ciascuno di essi.

```

035. //-----
036. // INTERFACE
037. //-----
038.
039. public boolean setNome(char allome)
040. {
041.     boolean wasSet = false;
042.     nome = allome;
043.     wasSet = true;
044.     return wasSet;
045. }
046.
047. public boolean setAscissa(int aascissa)
048. {
049.     boolean wasSet = false;
050.     ascissa = aascissa;
051.     wasSet = true;
052.     return wasSet;
053. }
054.
055. public boolean setOrdinata(int aordinata)
056. {
057.     boolean wasSet = false;
058.     ordinata = aordinata;
059.     wasSet = true;
060.     return wasSet;
061. }
062.
063. public boolean setModulo(float aModulo)
064. {
065.     boolean wasSet = false;
066.     modulo = aModulo;
067.     wasSet = true;
068.     return wasSet;
069. }

070. public boolean setAngolo(float aAngolo)
071. {
072.     boolean wasSet = false;
073.     angolo = aAngolo;
074.     wasSet = true;
075.     return wasSet;
076. }
077.
078.
079. public char getNome()
080. {
081.     return nome;
082. }
083.
084. public int getAscissa()
085. {
086.     return ascissa;
087. }
088.
089. public int getOrdinata()
090. {
091.     return ordinata;
092. }
093.
094. public float getModulo()
095. {
096.     return modulo;
097. }
098.
099. public float getAngolo()
100. {
101.     return angolo;
102. }
```

Infine viene scritto il prototipo di un metodo `delete()` e un metodo per visualizzare il contenuto di ogni attributo sullo schermo in formato stringa.

```

104.    public void delete()
105.    {
106.    }
107.
108.    public String toString()
109.    {
110.        String outputString = "";
111.        return super.toString() + "[" +
112.            "name" + ":" + getName() + "," +
113.            "ascissa" + ":" + getAscissa() + "," +
114.            "ordinata" + ":" + getOrdinata() + "," +
115.            "modulo" + ":" + getModulo() + "," +
116.            "angolo" + ":" + getAngolo() + "]";
117.    }
118.
119. }
```

Se ora selezioniamo dall'elenco **Java API Doc** e clicchiamo nuovamente su **[GENERATE]** viene creata anche la documentazione **JavaDOC** della nostra classe, che riportiamo di seguito.

The screenshot shows the Java API Doc interface with the following details:

- Navigation:** All Classes, Package, Class (highlighted), Tree, Deprecated, index, Help.
- Class Selection:** Punto
- Summary:**
  - Class Punto**
  - Inheritance: java.lang.Object → Punto
  - Code Snippet: 

```
public class Punto
extends java.lang.Object
```
  - Constructor Summary**
    - Constructors**
    - Constructor and Description**
    - Punto(char aNome, int aAscissa, int aOrdinata, float aModulo, float aAngolo)
  - Method Summary**
    - Methods**

Modifier and Type	Method and Description
void	delete()
float	getAngolo()
int	getAscissa()
float	getModulo()
char	getNome()
int	getOrdinata()



## Prova adesso!

- Definire una classe
- Utilizzare UMPLE

Realizza le seguenti classi:

- ▶ Contatore
- ▶ Quadrato
- ▶ Rettangolo
- ▶ Triangolo
- ▶ Cerchio
- ▶ Numero complesso

Quindi aggiungi anche i metodi e crea una gerarchia tra le classi, completando lo schema della figura seguente:



## ■ Conclusioni

Sono molte le possibilità offerte da **Umple**: vale la pena dedicare qualche ora per conoscere e valutare uno strumento molto ben documentato e facile da utilizzare che sicuramente porterà grosse soddisfazioni (e risparmio di tempo) a tutti i programmati non solo nella codifica dei linguaggi come **Java** e **C++** ma anche nella realizzazione di pagine **PHP** e nella creazione di basi di dati in linguaggio **SQL**.



VERSIONE  
SCARICABILE  
**EBOOK**

e-ISBN 978-88-203-6893-7

**[www.hoepliscuola.it](http://www.hoepliscuola.it)**

Ulrico Hoepli Editore S.p.A.  
via Hoepli, 5 - 20121 Milano  
e-mail [hoepliscuola@hoepli.it](mailto:hoepliscuola@hoepli.it)