

# Corso di Informatica

(Ivan Scagnetto – [scagnett@dimi.uniud.it](mailto:scagnett@dimi.uniud.it))

Laurea Specialistica in  
Traduzione e mediazione culturale.  
Lingue dell'Europa centrale e orientale

# Finalità del corso

- Fornire le nozioni informatiche di base al fine di:
  - comprendere il funzionamento del calcolatore e del software di base che “gira” su di esso;
  - apprendere il minimo necessario del gergo informatico per capire la documentazione tecnica;
  - essere in grado di sfruttare gli strumenti informatici per aumentare la propria produttività.

# Programma del corso

- Il corso è strutturato in tre parti:
  - parte I: nozioni sull'architettura degli elaboratori, sui sistemi operativi, rappresentazione dell'informazione, utilizzo di editor di testo e word processor (Microsoft Word);
  - parte II: documenti ipertestuali e multimediali (ipermedia), linguaggi markup (HTML, XML), strumenti per produrre ipertesti;
  - parte III: corpora linguistici elettronici: nozioni di base ed utilizzo.

# Bibliografia

- Libro di testo:

**Franco G.**

**“Corso di Informatica di Base”**

**Forum Editore, 2003, Udine**

- Libro di consultazione/integrazione:

*Elia A.*

*“Inchiostro Digitale – Tecnologie e scienze umane:  
scrivere, comunicare, insegnare con i nuovi media”*

*Gruppo Editoriale Esselibri-Simone, 2004*

# Orario di ricevimento

- **Mercoledì, 9.30-10.30**

Laboratorio di Elaborazione Dati

Dipartimento di Matematica e Informatica

Via delle Scienze, 206

Località Rizzi

- **e-mail:** [scagnett@dimi.uniud.it](mailto:scagnett@dimi.uniud.it)

# Informazioni e algoritmi

- L'ACM (Association for Computing Machinery), che annovera fra i suoi membri il maggior numero di operatori e scienziati del settore, dà la seguente definizione di Informatica:

*“L'informatica è la scienza degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione.”*

# Informatica

- Informatica: contrazione di “Informazione automatica”.
- I concetti fondamentali da tenere presenti sono due:
  - informazione (conoscenza, sapere)
  - algoritmo (sequenza di operazioni per la risoluzione di un problema)

# Algoritmi

- La parola algoritmo deriva dal matematico persiano **Al-Kuwarizmi**.
- Fondamentalmente è la descrizione rigorosa (ottenuta tramite un insieme **finito** di regole) di una sequenza di operazioni per la risoluzione di un problema.
- Quindi un algoritmo è una procedura meccanica per risolvere un problema in un numero finito di passi.

# Informazione

- Cos'è l'informazione?
  - **Shannon**: l'informazione dipende dal numero di alternative e dalla loro probabilità.
  - **Bateson**: acquisire informazioni significa percepire delle differenze.
- L'informazione contribuisce ad aumentare la nostra conoscenza (di fatti, cose ecc.).

# Informazioni vs. Dati

- La raccolta, l'archiviazione e la manipolazione di dati sono operazioni ricorrenti in molte attività (e.g., conti bancari, elenchi telefonici, elenchi degli iscritti ad un corso di laurea, elenco dei volumi di una biblioteca, catalogazione di reperti archeologici, dizionari ecc.).
- Tali attività possono prescindere dall'uso di un computer; questi ultimi garantiscono “solamente” una memorizzazione ed un trattamento dei dati stabili ed efficienti.
- Un dato in sé non costituisce un'informazione in quanto consiste semplicemente di un insieme di simboli; ad esempio la sequenza di caratteri *Mario Rossi* e le cifre *06 658976* non hanno un significato intrinseco.
- Quando un dato viene interpretato come risultato di un'interrogazione (e.g., “chi è il direttore della banca e qual è il suo numero telefonico?”) diventa informazione.
- Quindi un dato interpretato in un contesto può diventare informazione, ovvero, può arricchire la nostra conoscenza.

# Informazioni vs. Dati

- Per evidenziare la differenza concettuale fra informazioni e dati si può pensare alle seguenti analogie:
  - dati → sintassi → codifica (simboli)
  - informazioni → semantica → significato

# Rappresentazione dei numeri

- Tutte le persone, a partire dai primi anni di vita, possiedono il concetto di numero.
- Numero è un concetto astratto utile nelle operazioni di conteggio di oggetti di varia natura (cioè, di vari insiemi).
- Quindi un numero non è legato alla natura degli elementi contati, ma è un'astrazione di tutti gli insiemi aventi quel numero di elementi.

# Rappresentazione dei numeri

- Un numero non va confuso con la sua rappresentazione tramite simboli.
- Ad esempio il numero *tredici* viene rappresentato simbolicamente con **13**.
- La rappresentazione simbolica è necessaria per descrivere i numeri che non sono direttamente percepibili dal nostro sistema sensoriale (ad esempio **1.264.748**).

# I sistemi di numerazione

- I sistemi di numerazione sono nati con lo scopo di rappresentare simbolicamente i numeri.
- Ad esempio il numero *sette* può essere rappresentato come segue:

<b>IIII III</b>	Sistema arcaico
<b>VII</b>	Sistema romano
<b>7</b>	Sistema decimale
<b>111</b>	Sistema binario

# I sistemi di numerazione

- I sistemi di numerazione sono virtualmente infiniti.
- Tuttavia si è soliti fare un'importante distinzione:
  - i sistemi numerici come quello decimale o quello binario si dicono **posizionali** in quanto i simboli della rappresentazione (le cifre) acquistano significato diverso in base alla loro posizione;
  - i sistemi numerici come quello romano sono invece **non posizionali** in quanto ogni simbolo rappresenta sempre un ben preciso numero, indipendentemente dalla posizione in cui compare.

# Il sistema di numerazione romano

- Simboli e corrispondenze con il sistema decimale:

<b>I</b>	<b>II</b>	<b>III</b>	<b>IV</b>	<b>V</b>	<b>X</b>	<b>L</b>	<b>C</b>	<b>D</b>	<b>M</b>
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>10</b>	<b>50</b>	<b>100</b>	<b>500</b>	<b>1000</b>

- La rappresentazione di un numero è fondata su un principio additivo:

**sette = VII = V + I + I = cinque + uno + uno**

- Bisogna utilizzare sempre i valori più grandi possibili per rappresentare un numero:

**quindici = XV (e non VVV)**

# Il sistema di numerazione romano

- Per evitare di dover usare molti simboli per rappresentare numeri grandi, fu introdotta la notazione sottrattiva:

**cinquantanove = LIX = L + (X - I) =  
cinquanta + (dieci - uno) =  
cinquanta + nove**

invece di scrivere

**cinquantanove = LVIII**

# Il sistema di numerazione romano

- Le regole per la notazione sottrattiva sono:
  - gli unici simboli che possono essere usati in modo sottrattivo sono **I**, **X** e **C**;
  - soltanto un singolo simbolo rappresentante un numero più piccolo può essere messo a sinistra (ad esempio: **diciannove = XIX = dieci + nove**, ma **diciotto** non può essere scritto come **XIIX**);
  - Il numero da sottrarre deve essere maggiore o uguale a un decimo del numero da cui viene sottratto (quindi **quarantanove** si scrive **XLIX** e non **IL**).

# Svantaggi dei sistemi additivi

- I sistemi additivi (anche quelli che ricorrono alla notazione sottrattiva) presentano due grossi problemi:
  - l'elevato numero di simboli necessari per rappresentare numeri grandi,
  - la difficoltà di eseguire delle operazioni di calcolo anche banali (come addizioni, sottrazioni, moltiplicazioni).

# La notazione posizionale

- La notazione posizionale fu introdotta dai Caldei e dai Babilonesi.
- Si basa sul principio che ogni simbolo assume un significato diverso a seconda della posizione (casella) in cui compare.
- Nel sistema numerico decimale, ad esempio, abbiamo che nella rappresentazione **525** il **5** più a sinistra rappresenta il numero *cinquecento*, mentre il **5** più a destra rappresenta il numero *cinque*.

# La notazione posizionale

- Nella rappresentazione decimale **525** il **5** più a sinistra compare nella posizione (casella) delle **centinaia**, mentre il **5** più a destra compare nella posizione (casella) delle **unità**.

$$\bullet \quad 525 = \boxed{5} \times 100 + \boxed{2} \times 10 + \boxed{5} \times 1$$

↓                      ↓                      ↓

**centinaia      decine      unità**

# Sistema di numerazione in base dieci

- La notazione decimale (che utilizziamo quotidianamente) è anche chiamata sistema di numerazione in base dieci.
- Il motivo della denominazione è che il peso dei simboli è dato da potenze di dieci:
  - **unità:**  $10^0 = 1$
  - **decine:**  $10^1 = 10$
  - **centinaia:**  $10^2 = 100$
  - **migliaia:**  $10^3 = 1000$
  - ...

# Sistemi di numerazione posizionali

- Si dice che la notazione posizionale è universale per sottolineare che il principio su cui si fonda è indipendente dalla base.
- Quindi, oltre alla base dieci, possiamo scrivere dei numeri in base otto ad esempio: nella rappresentazione **525** in base otto, il **5** più a sinistra “pesa”  $8^2=64$ , mentre il **5** più a destra “pesa”  $8^0=1$ .

# Convenzione

- Se non è evidente dal contesto, è opportuno denotare la base del sistema di numerazione mettendola a pedice della rappresentazione numerica:
  - $525_{10}$  significa la rappresentazione decimale  $525$ ;
  - $525_8$  significa la rappresentazione ottale (ovvero, in base otto)  $525$ .

# Sistemi di numerazione posizionali

- Per definire un sistema di numerazione posizionale in una qualche base, occorre fissare quanto segue:
  - il valore numerico della base **b**;
  - i simboli (cifre) utilizzabili nelle rappresentazioni:  
 $\{s_1, s_2, \dots, s_b\}$
- Ad esempio, per la base dieci avremo quanto segue:
  - valore numerico della base **b = 10**;
  - simboli (cifre) utilizzabili nelle rappresentazioni:  
 $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

# Sistemi di numerazione posizionali

- Per il sistema ottale (ovvero, in base otto) avremo quanto segue:
  - valore numerico della base  **$b = 8$** ;
  - simboli (cifre) utilizzabili nelle rappresentazioni:  
 **$\{0, 1, 2, 3, 4, 5, 6, 7\}$**
- Come si rappresenta il numero otto nel sistema ottale?

$$\text{otto} = 10_8 = 1 \times 8^1 + 0 \times 8^0 = 8_{10}$$

# Sistemi di numerazione posizionali in informatica

- Nel mondo dell'informatica sono molto utilizzate le basi **2**, **8** e **16** (sistemi **binario**, **ottale** e **esadecimale**).
  - sistema binario: **base = 2**, cifre: **{0, 1}**;
  - sistema ottale: **base = 8**, cifre: **{0, 1, 2, 3, 4, 5, 6, 7}**;
  - sistema esadecimale: **base = 16**, cifre: **{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}**.

# Il sistema esadecimale

- Siccome non c'erano simboli conosciuti per rappresentare i numeri da dieci a quindici con una singola cifra, si scelse di utilizzare le prime sei lettere dell'alfabeto:

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>

- Esempio: il numero  $229_{10}$  in base sedici si scrive **E5** poiché vale quanto segue:

$$\begin{aligned} \mathbf{E5}_{16} &= \mathbf{E} \times 16^1 + \mathbf{5} \times 16^0 \\ &= \mathbf{14} \times \mathbf{16} + \mathbf{5} \times \mathbf{1} \\ &= \mathbf{224} + \mathbf{5} \\ &= \mathbf{229}_{10} \end{aligned}$$

- Le rappresentazioni esadecimali vengono spesso denotate con il prefisso **0x** oppure **\x**. Quindi  $\mathbf{E5}_{16} = \mathbf{0xE5} = \mathbf{\xE5}$

# Il sistema binario

- Il sistema binario utilizza soltanto due cifre: **0** e **1**.
- Perché è importante il sistema binario?
  - È il minimo sistema in grado di codificare una differenza (**0** ≠ **1**).
  - È molto semplice rappresentare **0** e **1** elettronicamente tramite due tensioni anche molto basse.

# Il sistema binario

- Perché sarebbe problematico rappresentare elettronicamente più cifre (seguendo l'esempio del caso binario)?
  - Occorrerebbero tensioni elevate per poterle distinguere con i seguenti problemi:
    - consumo energetico;
    - dissipazione del calore;
    - affidabilità dei dispositivi;
    - velocità di funzionamento.

# bit

- Le cifre del sistema binario sono dette bit (acronimo di binary digit).
- 8 bit formano un byte.
- Quanti numeri possiamo rappresentare con n bit?
  - Con 1 bit, rappresentiamo i numeri **0** e **1**.
  - Con 2 bit abbiamo le seguenti rappresentazioni:

$00_2$	$0_{10}$
$01_2$	$1_{10}$
$10_2$	$2_{10}$
$11_2$	$3_{10}$

Numeri da **0** a **3**

# bit

- Con tre bit abbiamo le seguenti configurazioni:

$000_2$	$0_{10}$
$001_2$	$1_{10}$
$010_2$	$2_{10}$
$011_2$	$3_{10}$

$100_2$	$4_{10}$
$101_2$	$5_{10}$
$110_2$	$6_{10}$
$111_2$	$7_{10}$

Numeri da **0** a **7**

# bit

- Con quattro bit abbiamo le seguenti configurazioni:

$0000_2$	$0_{10}$
$0001_2$	$1_{10}$
$0010_2$	$2_{10}$
$0011_2$	$3_{10}$
$0100_2$	$4_{10}$
$0101_2$	$5_{10}$
$0110_2$	$6_{10}$
$0111_2$	$7_{10}$

$1000_2$	$8_{10}$
$1001_2$	$9_{10}$
$1010_2$	$10_{10}$
$1011_2$	$11_{10}$
$1100_2$	$12_{10}$
$1101_2$	$13_{10}$
$1110_2$	$14_{10}$
$1111_2$	$15_{10}$

Numeri da **0** a **15**



# bit

- Possiamo quindi dedurre che:
  - con **1** bit possiamo rappresentare **2** ( $2^1$ ) numeri diversi;
  - con **2** bit possiamo rappresentare **4** ( $2^2$ ) numeri diversi;
  - con **3** bit possiamo rappresentare **8** ( $2^3$ ) numeri diversi;
  - con **4** bit possiamo rappresentare **16** ( $2^4$ ) numeri diversi;
  - ...
  - con **n** bit possiamo rappresentare  **$2^n$**  numeri diversi.
- In generale in un sistema posizionale con base **b**, possiamo rappresentare  **$b^n$**  numeri diversi utilizzando **n** cifre.



# bit

- Dato che con **3 bit** si possono rappresentare **8** numeri diversi, mentre con **3 cifre decimali** si possono rappresentare **1.000 (=10<sup>3</sup>)** numeri diversi, si potrebbe essere indotti a pensare che con il sistema binario occorranza troppe cifre (bit) per rappresentare numeri grandi.
- Tuttavia ciò non è vero, dato che la quantità di numeri rappresentabili con **n** cifre cresce in modo **esponenziale** con il crescere di **n**.

# bit

<b>2 cifre decimali</b>	<b><math>10^2</math></b>	<b>=</b>	<b>100</b>
<b>4 bit</b>	<b><math>2^4</math></b>	<b>=</b>	<b>16</b>
<b>1 byte (8 bit)</b>	<b><math>2^8</math></b>	<b>=</b>	<b>256</b>
<b>2 byte</b>	<b><math>2^{16}</math></b>	<b>=</b>	<b>65.536</b>
<b>6 cifre decimali</b>	<b><math>10^6</math></b>	<b>=</b>	<b>1.000.000</b>
<b>4 byte</b>	<b><math>2^{32}</math></b>	<b>=</b>	<b>4.294.967.296</b>

- Quindi con **32 bit** possiamo rappresentare più di **4 miliardi** di numeri diversi ed aggiungendo un solo bit, possiamo rappresentarne il doppio.

# bit

- Quanti bit sono necessari per rappresentare **m** numeri diversi?
  - La risposta è il più piccolo valore **n** tale che  $2^n \geq m$ .
- Se **n** è tale che  $2^n = m$ , allora **n** viene detto il logaritmo in base **2** di **m** e si indica con  $\log_2 m$ .
- Se  $\log_2 m$  non è un valore intero, si prende il valore intero immediatamente superiore (denotato con  $\lceil \log_2 m \rceil$ ), dato che non avrebbe senso considerare un numero frazionario di bit per rappresentare dei numeri nel sistema binario.

# Esempio

- Quanti bit servono per rappresentare **20** numeri diversi?
- Abbiamo che  $\log_2 20 = 4,32\dots$ , quindi  $\lceil \log_2 20 \rceil = 5$
- Se ne deduce che per rappresentare **20** numeri diversi servono **5 bit**. Infatti con **4 bit** possiamo rappresentare  $2^4=16$  numeri diversi (troppo pochi), mentre con **6 bit** possiamo rappresentare  $2^6=64$  numeri diversi (troppi).

# Cambiamento di base

- Dato che ci sono parecchi sistemi di numerazione posizionali, sorge il problema di come passare da una rappresentazione simbolica all'altra (cambiamento o passaggio di base).
- Categorie di cambiamenti di base:
  - da una base  $b$  a base 10 e viceversa;
  - da una base  $b$  a una base  $b'$  sua potenza e viceversa;
  - da una generica base  $b$  ad un'altra generica base  $b'$ , effettuando il passaggio intermedio alla base 10.

# Passaggio da una base $b$ alla base 10

- Per convertire in base 10 il numero rappresentato in base  $b$  da  $(c_n c_{n-1} \dots c_0)_b$ , è sufficiente eseguire il calcolo seguente:

$$(c_n \times b^n + c_{n-1} \times b^{n-1} + \dots + c_1 \times b^1 + c_0 \times b^0)_{10}$$

- Ad esempio:

$$\begin{aligned}(3765)_8 &= (3 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 5 \times 8^0)_{10} \\ &= (3 \times 512 + 7 \times 64 + 6 \times 8 + 5 \times 1)_{10} \\ &= (1536 + 448 + 48 + 5)_{10} = (2037)_{10}\end{aligned}$$

# Passaggio da base 10 ad una base b

- Per passare da una rappresentazione in **base 10** ad una in **base b**, è sufficiente dividere successivamente (finché si ottiene zero come risultato) il numero di partenza per la base b e leggere i resti delle divisioni al contrario.
- Esempio: convertire  $(2037)_{10}$  in base 8

2037		5
254		6
31		7
3		3
0		



$$(2037)_{10} = (3765)_8$$

# Passaggio da una base $b$ ad una $b'$

- Si effettua con il passaggio intermedio alla **base 10**.
- Esempio: convertire  $(3765)_8$  in **base 16**
  - Abbiamo già calcolato che  $(3765)_8 = (2037)_{10}$ .
  - Convertiamo  $(2037)_{10}$  in **base 16**:

<b>2037</b>		<b>5</b>		
<b>127</b>		<b>F</b>	←	<b>15</b>
<b>7</b>		<b>7</b>		
<b>0</b>				

$(2037)_{10} = (7F5)_{16}$

# Passaggio da una base $b$ a una base $b'$ sua potenza e viceversa

- I casi di passaggio da una base  $b$  ad una  $b'$  in cui una è potenza dell'altra sono i più facili.
- Esempio: passare da base **2** a base **8** ( $= 2^3$ )
  - è sufficiente dividere (a partire da destra) a gruppi di tre i bit che compongono la rappresentazione binaria e convertire in cifra ottale ogni singolo gruppetto di bit;
  - infatti con **3 bit** possiamo rappresentare **8 numeri** diversi **da 0 a 7**.

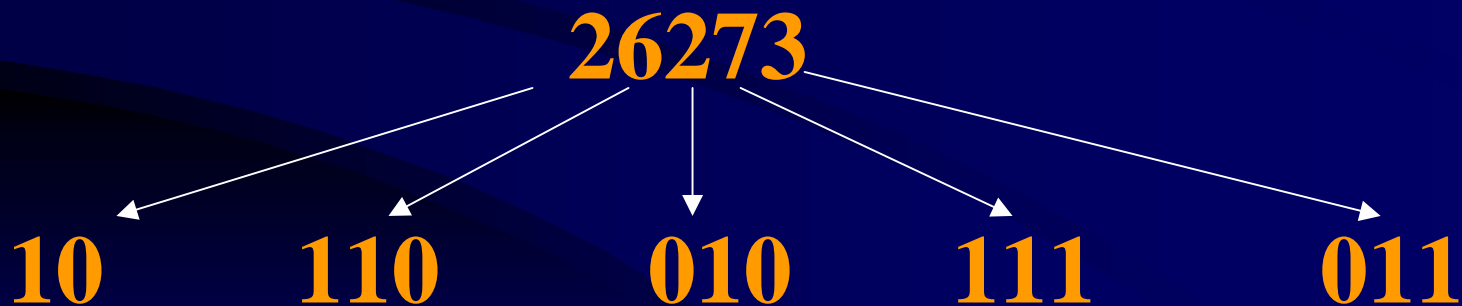
**10110010111011** → **10 110 010 111 011**  

---

**2 6 2 7 3**

# Passaggio da una base $b$ a una base $b'$ sua potenza e viceversa

- Esempio: passaggio inverso da base  $8 (= 2^3)$  a base  $2$ 
  - è sufficiente espandere ogni singola cifra ottale nella corrispondente rappresentazione binaria.



$$(26273)_8 = (10110010111011)_2$$

# Passaggio da una base $b$ a una base $b'$ sua potenza e viceversa

- Esempio: passaggio da base **2** a base **16** ( $= 2^4$ )
  - è sufficiente dividere (a partire da destra) a gruppi di quattro i bit che compongono la rappresentazione binaria e convertire in cifra esadecimale ogni singolo gruppetto di bit;
  - infatti con **4 bit** possiamo rappresentare **16 numeri** diversi **da 0 a 15**.

**10110010111011** → **10 1100 1011 1011**

---

**2 C B B**

# Passaggio da una base $b$ a una base $b'$ sua potenza e viceversa

- Esempio: passaggio inverso da base **16** ( $= 2^4$ ) a base **2**
  - è sufficiente espandere ogni singola cifra esadecimale nella corrispondente rappresentazione binaria.



$$(2CBB)_{16} = (10110010111011)_2$$

# Aritmetica

- Il principale vantaggio di un sistema posizionale rispetto ad altri sistemi di numerazione (come quello romano) è dato dal seguente fatto:
  - ogni numero può essere rappresentato utilizzando soltanto un numero ben definito di cifre diverse.
- Quindi le regole per il calcolo numerico possono essere descritte tramite semplici tabelle di addizione e moltiplicazione (da imparare a memoria).

# Addizioni

- Ad esempio è molto facile calcolare  $1542 + 841$ , conoscendo la tavola di addizione del sistema decimale:

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

$$\begin{array}{r} 1542 + \\ 1841 = \\ \hline \end{array}$$

$$2383$$

$$5 + 8 = 3 \text{ (con riporto di 1)}$$

# Addizioni

- Eseguire le addizioni di numeri anche molto grandi diventa quindi banale:
  - è sufficiente sommare le singole cifre in posizioni corrispondenti, andando a leggere la tavola dell'addizione (incrociando la riga e la colonna relative alle cifre in gioco);
  - nel caso la somma abbia come risultato un numero a due cifre, si effettua il riporto di quella più a sinistra.
- Abbiamo quindi una **procedura meccanica** che ci consente in un **numero finito di passi** di effettuare delle somme (**algoritmo di addizione**).

# Addizioni binarie

- Nel caso del sistema binario la tavola dell'addizione è molto semplice:

+	0	1
0	0	1
1	1	10

$$\begin{array}{r} 101101 + \\ \quad 1101 = \\ \hline 111010 \end{array} \quad \begin{array}{l} (45)_{10} \\ (13)_{10} \\ (58)_{10} \end{array}$$

# Moltiplicazioni

- La moltiplicazione  $m \times n$  di due numeri  $m$  e  $n$ , può essere calcolata per mezzo dell'addizione  $m + m + \dots + m$  (per  $n$  volte).
- Tuttavia esiste un algoritmo più veloce basato sulla tavola della moltiplicazione.

# Moltiplicazioni

- La tavola della moltiplicazione in base 10 è la seguente:

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

# Moltiplicazioni

- Esempio in base 10:

$$\begin{array}{r} 2378 \times \\ 25 = \\ \hline 11890 \\ 47560 \\ \hline 59450 \end{array}$$

$$\begin{aligned} 2378 \times 25 &= 2378 \times (20 + 5) \\ &= 2378 \times 20 + 2378 \times 5 \\ &= \boxed{2378 \times 2} \times \boxed{10} + \boxed{2378 \times 5} \end{aligned}$$

# Moltiplicazioni binarie

- Nel sistema binario la tavola della moltiplicazione è molto semplice:

$\times$	0	1
0	0	0
1	0	1

- Esempio:  
$$\begin{array}{r} 1011 \times \\ 100 = \\ \hline 0000 \\ 0000 \\ 1011 \\ \hline 101100 \end{array}$$
$$\begin{array}{r} (11)_{10} \\ (4)_{10} \\ 1011 \times 0 \\ 1011 \times 0 \\ 1011 \times 1 \\ (44)_{10} \end{array}$$

# Aritmetica al calcolatore

- Il presupposto principale da tenere a mente è che la memoria del calcolatore è composta da un insieme di “posizioni” (celle) con capacità limitata.
- Quindi è possibile memorizzare soltanto valori con un certo numero di cifre.
- Quanto detto si riassume dicendo che i numeri sono rappresentabili nel calcolatore con **precisione finita**.

# Aritmetica al calcolatore

- Esempio: con due cifre decimali possiamo rappresentare i numeri da 0 a 99.
- Con due cifre decimali non si possono rappresentare:
  - numeri negativi (-1, -2, -3, ...)
  - numeri maggiori di 99 (100, 101, 102, ...)
  - numeri razionali (frazioni)
  - numeri irrazionali (ad esempio  $\pi$ )

# Overflow

- Il fatto che i numeri siano rappresentabili con precisione finita ha delle conseguenze importanti per quanto riguarda l'aritmetica al calcolatore.
- Supponiamo ad esempio che vi siano soltanto 3 bit a disposizione per la rappresentazione di un numero e di voler eseguire la somma di 111 e 001:

$$\begin{array}{r} 111 + \\ 001 = \\ \hline 1000 \end{array}$$

The diagram shows a binary addition of 111 and 001. The result is 1000. The first bit of the result, '1', is circled in red, and a red arrow points from it to the right, indicating that this bit is the overflow.

Si rende necessario un **quarto bit** per rappresentare il risultato: **errore di overflow**.

# Limitazioni intrinseche

- Il problema della rappresentazione di valori numerici negativi può essere risolto.
- Il problema degli errori di overflow non ha una soluzione: è una limitazione intrinseca della rappresentazione dei numeri con precisione finita.

# Operazioni necessarie

- In realtà le uniche operazioni necessarie sono sommare uno (+1) e sottrarre uno (-1), unitamente alla capacità di contare quante volte viene eseguita un'operazione.
- Infatti:
  - la somma di  $m$  e  $n$  ( $m+n$ ) è uguale a sommare uno a  $m$  per  $n$  volte;
  - la sottrazione di  $n$  da  $m$  ( $m-n$ ) è uguale a sottrarre uno a  $m$  per  $n$  volte;
  - moltiplicazioni e divisioni possono essere eseguite tramite addizioni e sottrazioni rispettivamente (sempre contando il numero di applicazioni di un'operazione).

# Rappresentazione di numeri negativi

- Per risolvere il problema della rappresentazione dei numeri negativi vi sono vari approcci:
  - rappresentazione con bit di segno;
  - interpretazione (nel caso di n bit) eccesso  $2^{n-1}$ ;
  - notazione in complemento a uno;
  - notazione in complemento a due.

# Rappresentazione con bit di segno

- Supponiamo di avere a disposizione 8 bit; dividiamo l'intervallo dei 256 valori rappresentabili in due parti:
  - 0-127 (rappresentanti i numeri da 0 a 127);
  - 128-255 (rappresentanti i numeri da  $-0$  a  $-127$ ).
- Il bit più significativo (più a sinistra) rappresenta quindi il segno del numero:
  - 0: numero positivo (intervallo da 0 a 127);
  - 1: numero negativo (intervallo da 128 a 255).
- Aspetto negativo: lo zero ha due rappresentazioni:
  - $+0$  (00000000),
  - $-0$  (10000000).

# Interpretazione eccesso $2^{n-1}$

- Per evitare la doppia rappresentazione dello zero, una rappresentazione binaria viene interpretata come un numero a cui è stato aggiunto l'eccesso  $2^{n-1}$ .
- Esempio: consideriamo 8 bit per la rappresentazione.

Rappr. Binaria	Valore	Interp. eccesso 128
00000000	0	-128
00001000	8	-120
10000000	128	0
10000011	131	3

# Complemento a uno

- Rispetto alla rappresentazione con bit di segno abbiamo che:
  - se il bit di segno è 1 (numero negativo), tutti i bit restanti subiscono l'operazione di complemento a 1, ovvero, ogni 1 viene mutato in 0 e viceversa.

Rappr. binaria	Valore	Segno	7 bit	Compl.	7 bit	Valore rappr.
00001010	10	+	0001010	No	0001010	+10
10101101	173	-	0101101	Sì	1010010	-82
11111111	255	-	1111111	Sì	0000000	-0
00000000	0	+	0000000	No	0000000	+0

# Complemento a uno

- Svantaggi: doppia rappresentazione dello zero.
- Vantaggi: è facile fare le sottrazioni.  $m-n$  viene eseguito come la somma  $m + (-n)$ .
- Esempio: eseguiamo la sottrazione  $100 - 50 = 50$  (supponiamo di avere a disposizione 8 bit).
  - 100 si rappresenta in notazione in complemento a 1 come 01100100;
  - 50 si rappresenta in complemento a 1 come 00110010;
  - quindi  $-50$  si rappresenta in complemento a 1 come 11001101.

$$\begin{array}{r} 01100100 + \\ 11001101 = \\ \hline \end{array}$$



Sommando il riporto:

$$\begin{array}{r} 00110001 + \\ 1 = \\ \hline \end{array}$$

$$1\ 00110001$$

$$(50)_{10} = 00110010$$

# Complemento a due

- Per evitare il problema di sommare l'eventuale riporto oltre l'ultimo bit (l'ottavo nel caso dell'esempio) si è introdotta la notazione in complemento a due.
- Nella notazione in complemento a due infatti i numeri vengono rappresentati come nella notazione in complemento a uno tranne per il fatto che:
  - per i numeri negativi viene sommato 1 alla rappresentazione in complemento a uno.

# Complemento a due

- Esempio: eseguiamo la sottrazione  $100 - 50 = 50$  (supponiamo di avere a disposizione 8 bit).
  - 100 si rappresenta in notazione in complemento a 2 come 01100100;
  - 50 si rappresenta in complemento a 2 come 00110010;
  - quindi  $-50$  si rappresenta in complemento a 2 come  $11001101 + 1 = 11001110$ .

Il riporto oltre l'ottavo bit  
viene ignorato

$$\begin{array}{r} 01100100 + \\ 11001110 = \\ \hline \end{array}$$

→  $1\ 00110010 = (50)_{10}$