Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2021-2022

Codifica dell'informazione

Catia Prandi - catia.prandi2@unibo.it

Credit: Pietro Di Lena

There are 10 types of people in this world: those that understand binary and those that don't.

Introduzione

- Gli algoritmi sono costituiti da **istruzioni** che operano su **dati**.
- Per poter eseguire un programma su un calcolatore è necessario **rappresentare** i dati e le istruzioni in un formato che permetta al calcolatore di memorizzarli e manipolarli.
- Rappresentazione dell'informazione:
  - **Alfabeto**: set finito di simboli (segni grafici, colori, tensione elettrica, ...).
  - **Sintassi**: regole per codificare sequenze *ben formate* di simboli sull'alfabeto.
  - **Semantica**: regole per associare un significato alle sequenze di simboli.
- Esempio: il semaforo.
  - Alfabeto: colori `{rosso, giallo, verde}`
  - Sintassi: `verde` seguito da `giallo`, seguito da `rosso`, seguito da `verde`.
  - Semantica:
    - `verde` $\Longrightarrow$ "via libera",
    - `giallo` $\Longrightarrow$ "imminente cambio di stato",
    - `rosso` $\Longrightarrow$ "stop".
- Quale sistema di codifica è più adatto a rappresentare l'informazione su un calcolatore?
  - Limitazioni/vincoli: il calcolatore ha una **memoria limitata**.
  - Scelta progettuale: codifichiamo istruzioni e dati utilizzando lo stesso alfabeto.

Sistemi di numerazione

Quale di questi sistemi numerici è il più adatto per il calcolo automatico?

- **Sistema unario**. Sistema di numerazione additivo elementare in cui tutti i numeri interi sono rappresentati utilizzando esclusivamente un unico simbolo.

$$\boxed{|||||||||||\Longrightarrow 1+1+1+1+1+1+1+1+1+1+1=11}$$

  - Primitivo sistema di numerazione adottato dall'uomo.
  - Poco maneggevole in termini di utilizzo di memoria e calcolo numerico.
- **Sistemi additivi**. Sistema di numerazione basato su una legge additiva applicata ai simboli del sistema.

$$\boxed{\mathtt{XI}\Longrightarrow 10+1=1}$$

  - Sistemi additivi più noti: sistema romano, egizio e attico.
  - Più compatti del sistema additivo unario ma poco adatti per il calcolo.
- **Sistemi posizionali**. Sistema di numerazione in cui i simboli assumono valori diversi a seconda della posizione che occupano nella sequenza.

$$\boxed{11\Longrightarrow 1\times 10^1+1\times 10^0=11}$$

  - Introdotti da matematici arabi e indiani. Molto diffuso a partire dal X secolo.
  - In grado di rappresentare numeri grandi con notazione compatta e particolarmente adatto ai calcoli numerici.

Il sistema binario

- La quasi totalità dei calcolatori elettronici utilizzano il **sistema binario** per la rappresentazione interna dell'informazione.
- Il sistema numerico binario è un sistema **posizionale** a **base 2**. Ovvero, l'alfabeto consiste di due soli simboli, solitamente indicati con **0** e **1**.
- Motivazione di carattere tecnologico: le caratteristiche fisiche dei dispositivi che costituiscono un calcolatore rendono particolarmente conveniente la rappresentazione di due soli stati:
  - due diversi livelli di tensione elettrica,
  - due direzioni di polarizzazione di una sostanza magnetizzabile,
  - due diversi livelli di intensità della luce,
  - ecc.
- Unità di misura dell'informazione: **bit** (dall'inglese **b**inary dig**it**), definita come la *quantità minima di informazione che serve a discernere tra due possibili eventi equiprobabili*.

Unità di misura per la memorizzazione dell'informazione

Simbolo	in bit	in Byte	in Pow 2
bit (b)	1	1/8	2^1 bit
byte (B)	8	1	2^8 bit
kilobyte (KB)	8.192	1.024	2^{10} byte
megabyte (MB)	8.388.608	1.048.576	2^{20} byte
gigabyte (GB)	8.589.934.592	1.073.741.824	2^{30} byte
terabyte (TB)	8.796.093.302.400	1.099.511.628.000	2^{40} byte
petabyte (PB)	9.007.199.254.740.992	1.125.899.906.842.624	2^{50} byte
exabyte (EB)	9.223.372.036.854.775.808	1.152.921.504.606.846.976	2^{60} byte

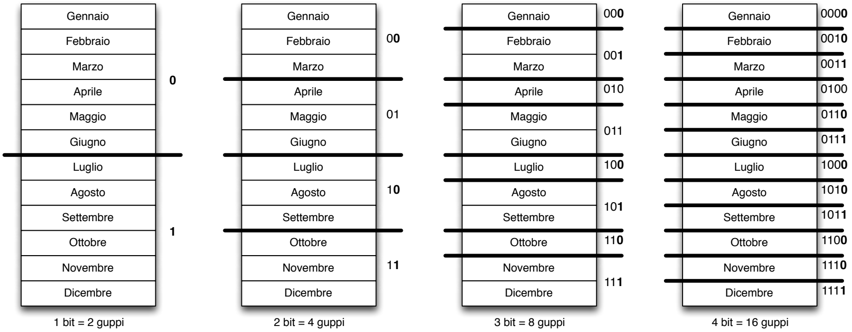
Codifica binaria dell'informazione

- **Problema**: assegnare un codice binario univoco ad un insieme predefinito di oggetti.
- Quanti oggetti possiamo codificare con n bit?
  - 1 bit: $2^1 = 2$ configurazioni (`0, 1`),
  - 2 bit: $2^2 = 4$ configurazioni (`00, 01, 10, 11`),
  - 3 bit: $2^3 = 8$ configurazioni (`000, 001, 010, 011, 100, 101, 110 , 111`),
  - n bit: 2^n configurazioni.
- **Attenzione**: assumiamo implicitamente che i codici abbiano tutti la stessa lunghezza di n bit (assunzione consistente con le caratteristiche di codifica su un calcolatore).
- Qual è il minimo numero n di bit sufficiente a codificare N oggetti distinti?

$$
\begin{aligned}
N &\leq 2^n \\
\log_2 N &\leq \log_2 2^n = n \log_2 2 \\
\log_2 N &\leq n \implies \\
n &= \lceil \log_2 N \rceil \text{ (intero superiore).}
\end{aligned}
$$

Esempio: codifica dei mesi dell'anno

- Per codificare 12 oggetti distinti abbiamo bisogno di $n = \lceil \log_2 12 \rceil = \lceil 3.58 \rceil = 4$ bit.



- Le configurazioni 0001, 0101, 1001 e 1101 sono inutilizzate.
- Ricordiamo che i codici devono avere tutti la stessa lunghezza.

Conversione delle codifiche: da binario a decimale

- Utilizziamo un sistema posizionale in base b.

$$\boxed{c_n c_{n-1} \ldots c_1 c_0 = c_n \times b^n + c_{n-1} \times b^{n-1} + \ldots + c_1 \times b^1 + c_0 \times b^0}$$

- **Conversione da binario a decimale**. Basta riscrivere il numero in notazione posizionale utilizzando la numerazione decimale.

$$\boxed{1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 = 12_{10}}$$

  Nota: la notazione 1100_2 indica il numero 1100 in base 2. La notazione 12_{10} indica il numero 12 in base 10.

- **Conversione da decimale a binario**. Basta riscrivere il numero in notazione posizionale utilizzando la numerazione binaria

$$\boxed{12_{10} = 1_2 \times 1010_2^1 + 10_2 \times 1010_2^0 = 1010_2 + 10_2 = 1100_2}$$

  Nota: 1010_2 è il numero 10 scritto in base due.

- La conversione da decimale a binario può essere effettuata in modo più semplice utilizzando il **metodo delle divisioni successive**.

Conversione da decimale a binario: metodo delle divisioni successive

Vogliamo convertire in base b in numero x in base 10. Consideriamo x descritto in notazione posizionale in base b (non conosciamo ancora tale rappresentazione).

$$\boxed{c_n \times b^n + c_{n-1} \times b^{n-1} + \ldots + c_1 \times b^1 + c_0 \times b^0}$$

1. Dividiamo il numero (divisione intera) per la base b

   $$\boxed{(c_n \times b^n + c_{n-1} \times b^{n-1} + \ldots + c_1 \times b^1 + c_0 \times b^0)/b}$$

   otteniamo

   - **quoziente**: $\boxed{c_n \times b^{n-1} + c_{n-1} \times b^{n-2} + \ldots + c_1}$
   - **resto**: $\boxed{c_0}$

2. Il resto corrisponde all'ultima cifra della rappresentazione in base b del numero.
3. Se il quoziente è uguale a zero, abbiamo finito e l'algoritmo termina.
4. Altrimenti, ripetiamo l'operazione al passo 1 sul quoziente, in modo da ottenere la cifra successiva.

Metodo delle divisioni successive: esempi

- Conversione del numero 12 da base 10 a base 2

				Quoziente	Resto
12	/	2	$\Longrightarrow$	6	**0**
6	/	2	$\Longrightarrow$	3	**0**
3	/	2	$\Longrightarrow$	1	**1**
1	/	2	$\Longrightarrow$	0	**1**

Numero 12 in base 2: 1100.

- Conversione del numero 12 da base 10 a base 3

				Quoziente	Resto
12	/	3	$\Longrightarrow$	4	**0**
4	/	3	$\Longrightarrow$	1	**1**
1	/	3	$\Longrightarrow$	0	**1**

Numero 12 in base 3: 110.

- Conversione del numero 12 da base 10 a base 8

				Quoziente	Resto
12	/	8	$\Longrightarrow$	1	**4**
1	/	8	$\Longrightarrow$	0	**1**

Numero 12 in base 8: 14.

Codifica ottale e esadecimale

- Le codifiche **ottale** e **esadecimale** sono spesso supportate dai linguaggi di programmazione in quanto molto utili per rappresentare in modo succinto i numeri in notazione binaria.
- **Codifica ottale**
  - Alfabeto ottale: $\{0, 1, 2, 3, 4, 5, 6, 7\}$

$$\boxed{175_8 = 1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 = 125_{10}}$$

  - Ogni cifra ottale corrisponde precisamente a tre cifre binarie

$$\boxed{01111101_2 = [01_2][111_2][101_2] = 175_8}$$

- **Codifica esadecimale**
  - Alfabeto esadecimale : $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

$$\boxed{7D_{16} = 7 \times 16^1 + 13 \times 16^0 = 125_{10}}$$

  - Ogni cifra esadecimale corrisponde precisamente a quattro cifre binarie

$$\boxed{01111101_2 = [0111_2][1101_2] = 7D_{16}}$$

Rappresentazione degli interi negativi

- Abbiamo visto come rappresentare un intero positivo in binario. Come rappresentiamo in binario gli **interi negativi**?
- Idea semplice: utilizziamo un bit per rappresentare il **segno** del numero.
  - Fissiamo il numero n di bit che sono utilizzati per rappresentare l'intero.
  - Utilizziamo 1 bit per il **segno**.
  - Utilizziamo $n-1$ bit per il **modulo**.

$$\boxed{+125_{10} = 01111101_2} \qquad \boxed{-125_{10} = 11111101_2}$$

- Assumendo di fissare $n = 8$ possiamo rappresentare come intero massimo il numero $2^7 - 1 = 127$ e come intero minimo $-2^7 + 1 = -127$.
- Problema: con questa codifica abbiamo due possibili rappresentazioni per lo zero!

$$\boxed{+0_{10} = 00000000_2} \qquad \boxed{-0_{10} = 10000000_2}$$

Complemento a due

- Il **complemento a due** è il metodo più diffuso per la rappresentazione dei numeri con segno in informatica.
- Anche in questo caso è necessario fissare il numero n di bit che verranno utilizzati per rappresentare l'intero.
- Algoritmi di codifica in complemento a due:
  1. L'intero x è rappresentato dalla codifica binaria dell'intero $2^n + x$.
  2. Per negare un intero x (positivo o negativo) applichiamo la codifica $\widetilde{x} + 1$, dove $\widetilde{x}$ è il *complemento ad 1* di x (tutti i bit sono invertiti).
- Il primo bit continua ad indicare il segno negativo (1) e positivo (0) del numero.
- Esempi. Per $n = 8$

$$\boxed{+125_{10} = 2^8 + 125 = 381_{10} \Longrightarrow 01111101_2}$$

$$\boxed{-125_{10} \Longrightarrow 10000010_2 + 1 = 10000011_2}$$

- Con il complemento a due è possibile rappresentare gli interi nel range da -2^{n-1} a $2^{n-1} - 1$ e lo 0 ha una sola rappresentazione.

$$\boxed{-128_{10} = 2^8 - 128 = 128_{10} \Longrightarrow 10000000_2}$$

Operazioni di calcolo aritmetico

- Le operazioni di addizione e sottrazione di due numeri rappresentati con il complemento a due non richiede particolari accorgimenti, anche quando i due operandi sono di segno opposto.
- Esempio di addizione.

$$\begin{array}{rc|r} 1\ 1111\ 111 & & \text{riporto} \\ 0000\ 1111 & + & 15_{10} \\ 1111\ 1011 & & -5_{10} \\ \hline 0000\ 1010 & & 10_{10} \end{array}$$

  Il riporto finale di 1 viene ignorato.
- Esempio di sottrazione.

$$\begin{array}{rc|r} 1\ 1111\ 000 & & \text{riporto} \\ 0000\ 1111 & - & 15_{10} \\ 1111\ 1011 & & -5_{10} \\ \hline 0001\ 0100 & & 20_{10} \end{array}$$

  Il riporto finale di 1 viene ignorato.

Operazioni di calcolo aritmetico: overflow

- Le regole per individuare situazioni di overflow con la rappresentazione di tipo complemento a due sono semplici:
  1. Se la somma di due numeri positivi genera un numero negativo, allora l'operazione è andata in overflow.
  2. Se la somma di due numeri negativi genera un numero positivo, allora l'operazione è andata in overflow.
  3. Negli altri casi, non ci sono problemi di overflow.
- Esempio di overflow durante una addizione.

$$\begin{array}{r|r} 1111\ 111 & \text{riporto} \\ 0111\ 1111\ + & 127_{10} \\ 0000\ 0001 & 1_{10} \\ \hline 1000\ 0000 & -128_{10} \end{array}$$

Il riporto finale cambia il segno dell'operazione.

Numeri frazionari in binario

- Come rappresentiamo un numero frazionario in binario o altra base?
  - Utilizziamo ancora una volta il sistema posizionale, questa volta considerando potenze negative della base per la parte frazionaria.
- Conversione di 0.101 da binario in base 10

$$\boxed{0.101_2 = 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 0 + 1/2 + 0 + 1/8 = 0.625_{10}}$$

- Conversione di 0.1 da binario in base 10

$$\boxed{0.1_2 = 0 \times 2^0 + 1 \times 2^{-1} = 0 + 1/2 = 0.5_{10}}$$

- Conversione di 0.1 da base tre in base 10

$$\boxed{0.1_3 = 0 \times 3^0 + 1 \times 3^{-1} = 0 + 1/3 = 0.\overline{3}_{10}}$$

- Conversione di 0.00011 da binario in base 10

$$\boxed{0.00011_2 = 1 \times 2^{-4} + 1 \times 2^{-5} = 1/16 + 1/32 = 0.09375_{10}}$$

- La conversione in base diversa da 10 è più complessa (richiede il calcolo di operazioni in base $b \neq 10$) ma esistono algoritmi alternativi più maneggevoli.

Algoritmo per convertire una frazione decimale in binario

Algoritmo per la conversione di un numero $0 \leq n < 1$ da base 10 in base b:

1. Moltiplichiamo il numero n per la base b
2. La parte intera di $n \times b$ è una cifra del numero frazionario in base b
3. Rimuoviamo da $n \times b$ la parte intera e ripartiamo dal punto 1.
4. Continuiamo fino a quando $n = 0$ oppure siamo in un ciclo

Algoritmo per convertire una frazione decimale in binario: esempi

- Conversione del numero 0.625 da base 10 a base 2

				Risultato	Parte intera
0.625	$\times$	2	$\Longrightarrow$	1.25	**1**
0.25	$\times$	2	$\Longrightarrow$	0.5	**0**
0.5	$\times$	2	$\Longrightarrow$	1	**1**

Numero 0.625 in base 2: **0.101**

- Conversione del numero 0.5 da base 10 a base 2

				Risultato	Parte intera
0.5	$\times$	2	$\Longrightarrow$	1.0	**1**

Numero 0.5 in base 2: 0.1.

- Conversione del numero 0.1 da base 10 a base 2

				Risultato	Parte intera
0.1	$\times$	2	$\Longrightarrow$	0.2	**0**
0.2	$\times$	2	$\Longrightarrow$	0.4	**0**
0.4	$\times$	2	$\Longrightarrow$	0.8	**0**
0.8	$\times$	2	$\Longrightarrow$	1.6	**1**
0.6	$\times$	2	$\Longrightarrow$	1.2	**1**
0.2	$\times$	2	$\Longrightarrow$	0.4	**0**

Numero 0.1 in base 2: $0.0\overline{0011}$ (non ha rappresentazione finita).

Rappresentazione dei numeri reali

- Come sono rappresentati sul calcolatore i numeri non interi?
  - Abbiamo la necessità di memorizzare sia la parte intera che la parte frazionaria del numero.
- Abbiamo due principali codifiche dei numeri frazionari: **virgola fissa** e **virgola mobile**.
- **Virgola fissa**. Una parte delle cifre della codifica è dedicata a rappresentare la parte intera e il resto a rappresentare la parte frazionaria.
  - Problemi: non rappresenta bene numeri frazionari molto grandi o molto piccoli
- **Virgola mobile** o **floating point**. Una parte delle cifre della codifica è dedicata a rappresentare un esponente della base che indica l'ordine di grandezza del numero.
  - A partità di cifre, estende l'intervallo di numeri rappresentabili in virgola mobile.
  - Fa uso della notazione esponenziale:

$$\boxed{-30.375 = -0.30375 \times 10^2 = -0.030375 \times 10^3 = -30375.0 \times 10^{-3}}$$

  - Principio della codifica **floatig point**: si fa *scorrere* la virgola decimale ad una posizione *conveniente*, utilizzando la notazione esponenziale.

Rappresentazione in virgola mobile (cenni)

- Lo standard IEEE 754 specifica il formato per la rappresentazione dei numeri in virgola mobile
  - Precisione *singola* a 32 bit

1	8	23
+/-	Esponente	Mantissa

  - Precisione *doppia* a 64 bit

1	11	52
+/-	Esponente	Mantissa

  - La base è implicita e non viene quindi rappresentatata.
- Dettagli: il numero n è rappresentato tramite una tripla (s, m, e)

$$n = (-1)^s \cdot m \cdot b^{\pm e}$$

  - m: **mantissa** (detta anche *significante*). Normalizzata tra due potenze successive della base. Convenzionalmente, la prima cifra significativa si trova immediatamente a sinistra del punto decimale. Es $\boxed{-30.375 \Rightarrow -3.0375 \times 10^1}$, $\boxed{10.11_2 \Rightarrow 1.011_2 \times 10_2^1}$
  - e: **esponente** (detto anche *caratteristica*) intero. Rappresentato in eccesso (*polarizzazione* o *bias*). Es. per la precisione singola, se gli 8 bit dell'esponente contengono 163_{10} allora l'esponente vale $\boxed{163_{10} - 127_{10} = 39_{10}}$ (bias 127).

Esempio: conversione da binario a decimale

- Consideriamo il seguente numero binario in precisione singola

1	10000001	10110011001100110011010

- **Segno**: 1. Il numero è negativo.
- **Esponente**: $10000001_2 = 129_{10}$. Dobbiamo sottrarre il *bias* 127. Quindi

$$e = 129 - 127 = 2.$$

- **Mantissa**: $10110011001100110011010_2 \Rightarrow 1.10110011001100110011010_2$ (ricordiamo che la mantissa è normalizzata). Conversione da binario a decimale

$$1.10110011001100110011010_2 = 1 \times 2^0 + 1 \times 2^{-1} + 02^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + ..$$
$$\simeq 1.7000000476837158_{10}$$

Quindi

$$m \simeq 1.7.$$

- Mettiamo tutto insieme

$$-1^1 \cdot m \cdot 2^e \simeq -1 \cdot 1.7 \cdot 2^2 = -6.8$$