

Programma del corso

□ *Introduzione agli algoritmi*

■ ***Rappresentazione delle Informazioni***

□ *Architettura del calcolatore*

□ *Elementi di Programmazione*

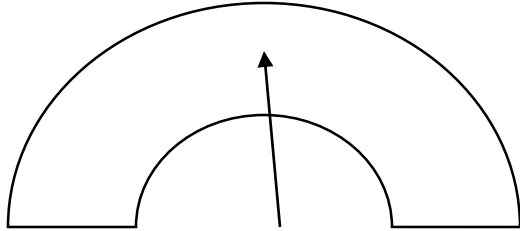
Rappresentazione dell'informazione

- Varie rappresentazioni sono possibili per la medesima informazione
 - Es. Testo scritto su carta o registrato su nastro

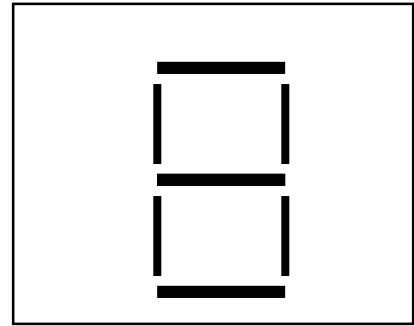
 - Rappresentazioni R1 e R2 sono equivalenti se data R1 è possibile ricavare R2 e viceversa
 - Es. Trascrizione del testo data la sua registrazione e viceversa

 - Scelta della rappresentazione
 - Spesso convenzionale ...
 - ... ma spesso legata a vincoli
 - Es. Rappresentazione binaria negli elaboratori
-

Analogico vs digitale



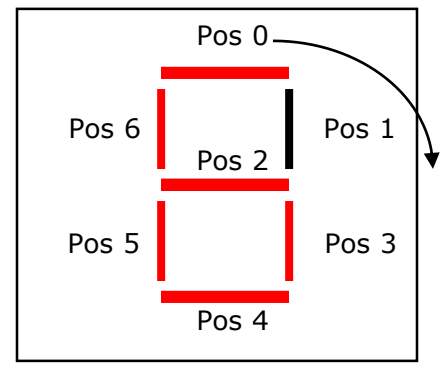
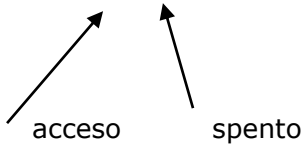
Informazione esplicita nel supporto: per analogia



Informazione implicita nella rappresentazione: serve codifica/decodifica

Il numero 6 si codificherebbe come

1011111





La rappresentazione digitale dell'informazione

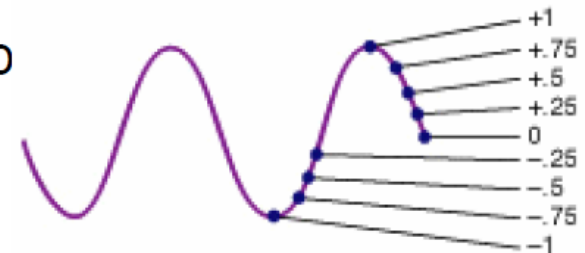
■ Rappresentazione digitale

- Ogni dato viene codificato impiegando entità distinte individualmente e organizzate in modo opportuno (es. abaco). Trova le sue origini nel conteggio con le *dita della mano* (da cui il nome).



■ Rappresentazione analogica

- Basata sull'impiego di dispositivi che realizzano una grandezza fisica che *può variare in modo continuo* (es. tensione elettrica).



■ Calcolatori analogici e digitali

- Durante questo secolo sono stati sviluppati sia calcolatori analogici che digitali ma, la rappresentazione che si è affermata è di tipo *digitale*.

Dalla rappresentazione alla codifica dell'informazione

rappresentazione

- Alfabeto: insieme di simboli
 - Es. le 10 cifre (da 0 a 9)
 - Stringhe: concatenazioni di simboli dell'alfabeto
 - Es. la stringa 123
 - Esiste un insieme di configurazioni possibili (di solito finito)
 - Processo di codifica: da informazione a una stringa che la rappresenta
 - Convenzionale: deve essere condiviso da chi usa
 - Processo di decodifica: da una stringa ad informazione
-

Codifica dell'informazione

- Il calcolatore memorizza ed elabora vari tipi di informazioni
 - Numeri, testi, immagini, suoni
 - Occorre rappresentare tale informazione in formato facilmente manipolabile dall'elaboratore
-



L'aritmetica dei calcolatori

- L'aritmetica usata dai calcolatori è diversa da quella comunemente utilizzata dalle persone.
- *La precisione* con cui i numeri possono essere espressi è *finita* e predeterminata poiché questi devono essere memorizzati entro un limitato spazio di memoria.

$$\sqrt{2} = \boxed{1 \ . \ 1 \ 4 \ 1 \ 4 \ 2} \ 1 \ 3 \ 5 \ 6$$

- *La rappresentazione* è normalmente ottenuta utilizzando il *sistema binario* poiché più adatto a essere maneggiato dal calcolatore.

$$124 \rightarrow 01111100$$

Numeri decimali	Numeri binari
0.....	0
1.....	1
2.....	1 0
3.....	1 1
4.....	1 0 0
5.....	1 0 1
6.....	1 1 0
7.....	1 1 1
8.....	1 0 0 0
9.....	1 0 0 1
10.....	1 0 1 0

Rappresentazione delle informazioni



Idea di fondo

- usare presenza/assenza di carica elettrica
- usare passaggio/non passaggio di corrente/luce

Usiamo cioè una rappresentazione binaria (a due valori) dell'informazione

L'unità minimale di rappresentazione è il **BIT** (**BI**nary digi**T** – cifra digitale): **0** o **1**

Informazioni complesse

Con 1 bit rappresentiamo solo 2 diverse informazioni:

si/no - on/off - 0/1

Mettendo insieme più bit possiamo rappresentare più informazioni:

00 / 01 / 10 / 11

Informazioni complesse si memorizzano come sequenze di bit

Informazioni complesse

- Per codificare i nomi delle 4 stagioni bastano 2 bit

 - Ad esempio:
 - **0 0** per rappresentare **Inverno**
 - **0 1** per rappresentare **Primavera**
 - **1 0** per rappresentare **Estate**
 - **1 1** per rappresentare **Autunno**

 - Quanti bit per codificare i nomi dei giorni della settimana?
-

Informazioni complesse

In generale, con **N** bit, ognuno dei quali può assumere **2** valori, possiamo rappresentare **2^N** informazioni diverse (**tutte le possibili combinazioni di 0 e 1 su N posizioni**)

viceversa

Per rappresentare **M** informazioni dobbiamo usare **N** bit, in modo che **$2^N \geq M$**

Esempio

Per rappresentare **57** informazioni diverse dobbiamo usare gruppi di almeno **6** bit. Infatti:

$$2^6 = 64 > 57$$

Cioè un gruppo di 6 bit può assumere 64 configurazioni diverse:

000000 / 000001 / 000010 ... / 111110 / 111111

Il Byte

□ Una sequenza di **8 bit** viene chiamata **Byte**

■ 0 0 0 0 0 0 0 0

■ 0 0 0 0 0 0 0 1

■

byte = 8 bit = $2^8 = 256$ informazioni diverse

Usato come unità di misura per indicare

- le dimensioni della memoria
- la velocità di trasmissione
- la "potenza" di un elaboratore

Usando sequenze di byte (e quindi di bit) si possono rappresentare caratteri, numeri, immagini, suoni.

Altre unità di misura

- ❑ KiloByte (**KB**), MegaByte (**MB**), GigaByte (**GB**)
 - ❑ Per ragioni storiche in informatica Kilo, Mega, e Giga indicano però le **potenze di 2** che più si avvicinano alle corrispondenti potenze di 10 (Sistema IEC)
 - ❑ Sistema SI: 1 Kilobyte = 1000 byte
 - ❑ Sistema IEC: 1 Kilobyte (detto *Kibibyte* = 1024 byte)

 - ❑ Più precisamente (sistema IEC)
 - 1 KB = 1024 x 1 byte = $2^{10} \sim 10^3$ byte
 - 1 MB = 1024 x 1 KB = $2^{20} \sim 10^6$ byte
 - 1 GB = 1024 x 1 MB = $2^{30} \sim 10^9$ byte

 - ❑ Il sistema IEC è usato come unità di misura per la capacità della memoria di un elaboratore.
 - ❑ Il sistema SI è usato come unità di misura per le capacità degli hard disk
-



Unità di misura nel sistema binario

- Il bit rappresenta la più piccola unità di misura dell'informazione memorizzabile in un calcolatore. I sistemi moderni memorizzano e manipolano miliardi di bit; per questo motivo sono stati definiti diversi multipli.

Nome	Sigla	In bit	In byte	In potenze di 2
Bit	Bit	1 bit	1/8	$2^1=2$ stati
Byte	Byte	8	1	$2^8=256$ stati
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	8.796.093.022.208	1.099.511.627.776	2^{40} Byte

ATTENZIONE: 1KB non corrisponde a 1000 Byte, ma a 1024 Byte, 1MB non corrisponde a 1000000 Byte, ...

Codici per i simboli dell'alfabeto

- Per rappresentare i simboli dell'alfabeto anglosassone (0 1 2 ... A B ... a b ...) bastano 7 bit (codifica **ASCII**)
 - Nota: *B* e *b* sono simboli diversi
 - 26 maiuscole + 26 minuscole + 10 cifre + 30 segni di interpunzione+... -> circa 120 oggetti

 - Per l'alfabeto esteso con simboli quali &, %, \$, ... bastano 8 bit come nella codifica accettata universalmente chiamata **ASCII esteso**

 - Per manipolare un numero maggiore di simboli si utilizza la codifica **UNICODE** a 16 bit
-

Codifica ASCII

- La codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nterchange **C**ode) utilizza codici su 7 bit
(**$2^7 = 128$ caratteri diversi**)
 - Ad esempio
 - 1 0 0 0 0 0 1 rappresenta A
 - 1 0 0 0 0 1 0 rappresenta B
 - 1 0 0 0 0 1 1 rappresenta C
 - Le parole si codificano utilizzando sequenze di valori da 7 bit
 - 1000010 1000001 1000010 1000001
 B A B A
-

Altri codici di codifica

□ **ASCII ESTESO**

- Usa anche il primo bit di ogni byte
- 256 caratteri diversi
- non è standard (cambia con la lingua usata)
- Ad es. a volte nello scambio di mail, ci si trova con strani caratteri (sono magari le lettere accentate non riconosciute dal programma di gestione delle mail)

□ **ISO 8859-1: contiene i caratteri latini di maggior uso (coincide con ASCII per i primi 127 valori)**

□ **UNICODE (UTF-8 e UTF-16)**

- standard proposto a 8 e 16 bit (65.536 caratteri)
- UTF-8 è usato per le e-mail

□ **EBCDIC**

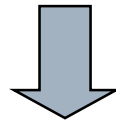
- altro codice a 8 bit della IBM (quasi in disuso)
-

Tabella ASCII (0-127)

00000000	Null	00100000	Spc	01000000	@	01100000	~
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	\$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	01000101	E	01100101	e
00000110	Acknowledge	00100110	&	01000110	F	01100110	f
00000111	Audible bell	00100111	'	01000111	G	01100111	g
00001000	Backspace	00101000	(01001000	H	01101000	h
00001001	Horizontal tab	00101001)	01001001	I	01101001	i
00001010	Line feed	00101010	*	01001010	J	01101010	j
00001011	Vertical tab	00101011	+	01001011	K	01101011	k
00001100	Form Feed	00101100	,	01001100	L	01101100	l
00001101	Carriage return	00101101	-	01001101	M	01101101	m
00001110	Shift out	00101110	.	01001110	N	01101110	n
00001111	Shift in	00101111	/	01001111	O	01101111	o
00010000	Data link escape	00110000	0	01010000	P	01110000	p
00010001	Device control 1	00110001	1	01010001	Q	01110001	q
00010010	Device control 2	00110010	2	01010010	R	01110010	r
00010011	Device control 3	00110011	3	01010011	S	01110011	s
00010100	Device control 4	00110100	4	01010100	T	01110100	t
00010101	Neg. acknowledge	00110101	5	01010101	U	01110101	u
00010110	Synchronous idle	00110110	6	01010110	V	01110110	v
00010111	End trans. block	00110111	7	01010111	W	01110111	w
00011000	Cancel	00111000	8	01011000	X	01111000	x
00011001	End of medium	00111001	9	01011001	Y	01111001	y
00011010	Substitution	00111010	:	01011010	Z	01111010	z
00011011	Escape	00111011	;	01011011	[01111011	{
00011100	File separator	00111100	<	01011100	\	01111100	
00011101	Group separator	00111101	=	01011101]	01111101	}
00011110	Record Separator	00111110	>	01011110	^	01111110	~
00011111	Unit separator	00111111	?	01011111	_	01111111	Del

Numeri in ASCII

Le cifre 0..9 rappresentate in Ascii sono simboli o caratteri **NON** quantità numeriche



Non possiamo usarle per indicare quantità e per le operazioni aritmetiche. (Anche nella vita di tutti giorni usiamo i numeri come simboli e non come quantità: i n. telefonici)

Il sistema decimale

- 10 cifre di base: 0, 1, 2, ..., 9
 - **Notazione posizionale:** la posizione di una cifra in un numero indica il suo **peso** in potenze di **10**. I pesi sono:
 - Unità = $10^0 = 1$ (posiz. 0-esima)
 - decine = $10^1 = 10$ (posiz. 1-esima)
 - centinaia = $10^2 = 100$ (posiz. 2-esima)
 - migliaia = $10^3 = 1000$ (posiz. 3-esima)
 -
-

Esempio di numero rappresentato in notazione decimale

Il **numerale** 2304 in notazione decimale (o in base 10) rappresenta la quantità:

$$2304 = 2*10^3 + 3*10^2 + 0*10^1 + 4*10^0 =$$

$$2000 + 300 + 0 + 4 = 2304 \text{ (**numero**)}$$

Nota: numero e numerale qui coincidono, perché il sistema decimale è quello adottato come sistema di riferimento

- NOTA: lo stesso numero è rappresentato da numerali diversi in diversi sistemi
 - **156** nel sistema decimale
 - **CLVI** in cifre romane
-

Notazione posizionale (decimale)

Dato un numerale espresso come:

□ $c_n c_{n-1} \dots c_1 c_0$

■ dove i coefficienti c_i possono essere le cifre da 0 a 9

Il numero corrispondente è:

□ $c_n * 10^n + c_{n-1} * 10^{n-1} + \dots + c_1 * 10^1 + c_0 * 10^0$

□ In base 10 con N cifre posso rappresentare i 10^N numeri da 0 a $10^N - 1$

Notazione posizionale (generale)

- Data una base B
 - considerato il numerale $c_n c_{n-1} \dots c_1 c_0$
 - dove i coefficienti c_i possono essere le cifre da 0 a B-1
 - Il numero corrispondente è:
 - $c_n * B^n + c_{n-1} * B^{n-1} + \dots + c_1 * B^1 + c_0 * B^0$
 - con N cifre posso rappresentare i B^N numeri da 0 a $B^N - 1$
-

Notazione posizionale (binaria)

□ Considerando $B=2$

□ Dato il numerale:

■ $c_n c_{n-1} \dots c_1 c_0$

■ dove i coefficienti c_i possono essere 0 o 1

■ Il numero è: $c_n * 2^n + \dots + c_2 * 2^2 + c_1 * 2^1 + c_0 * 2^0$

□ con N cifre riesco a rappresentare i 2^N numeri da 0 a $2^N - 1$

Il sistema binario

- 2 Cifre di base: 0 e 1
 - **Notazione posizionale:** la posizione di una cifra in un numero binario indica il suo **peso** in potenze di **2**. I pesi sono:
 - $2^0 = 1$ (posiz. 0-esima)
 - $2^1 = 2$ (posiz. 1-esima)
 - $2^2 = 4$ (posiz. 2-esima)
 - $2^3=8; 2^4=16; 2^5=32; 2^6=64; 2^7=128;$
 $2^8=256; 2^9=512; 2^{10} = 1024; 2^{11}=2048,$
 $2^{12}=4096; \dots$
-

Esempio di numero rappresentato in notazione binaria

Il **numerale** 10100101 in notazione binaria (o in base 2) rappresenta la quantità:

10100101

$$1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$

$$128 + 0 + 32 + 0 + 0 + 4 + 0 + 1 =$$

165 (**numero**)

Il numero più grande rappresentato con **N** cifre

□ Sist. Decimale = $99\dots99 = 10^N - 1$

□ Sist. Binario = $11\dots11 = 2^N - 1$

□ **Esempio:** 11111111 (8 bit binari) = $2^8 - 1 = 255$. Per rappresentare il n. 256 ci vuole un bit in più: 100000000 = $1 * 2^8 = 256$.

Quindi...

Fissate quante cifre (bit) sono usate per rappresentare i numeri, si fissa anche il numero più grande che si può rappresentare:

- con 16 bit: $2^{16} - 1 = 65.535$
 - con 32 bit: $2^{32} - 1 = 4.294.967.295$
 - con 64 bit: $2^{64} - 1 = \text{circa } 1,84 * 10^{19}$
-

Conversione da base 2 a base 10

Idea di fondo: usare le potenze di 2 che, sommate, danno il numero **N** da convertire:

- Prendere le potenze di $2 \leq$ di **N** nell'ordine dalla più grande alla più piccola (cioè 2^0)
 - Associare il bit 1 alle potenze che vengono usate nella somma per ricostruire **N**
 - Associare il bit 0 alle potenze non usate.
-

Conversione da base 2 a base 10

Basta moltiplicare ogni bit per il suo peso e sommare il tutto:

Esempio:

$$\begin{array}{r} 10100 \\ 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = \\ 16 + 4 = 20 \end{array}$$

la conversione e' una **somma di potenze**

(N.B. se il numero binario termina per 1 e' dispari altrimenti e' pari).

Conversione da base 10 a base 2

Regola:

- divido il numero per 2: il resto è la cifra c_0
 - divido il risultato per 2: il resto è la cifra c_1
 - divido il risultato per 2: il resto è la cifra c_2
 - mi fermo quando il risultato è 0
(eventualmente con resto 1)
-

Conversione da base 10 a base 2

□ Conversione di 29_{10}

$$29/2 = 14 \quad R = 1 (c_0)$$

$$14/2 = 7 \quad R = 0 (c_1)$$

$$7/2 = 3 \quad R = 1 (c_2)$$

$$3/2 = 1 \quad R = 1 (c_3)$$

$$1/2 = 0 \quad R = 1 (c_4)$$

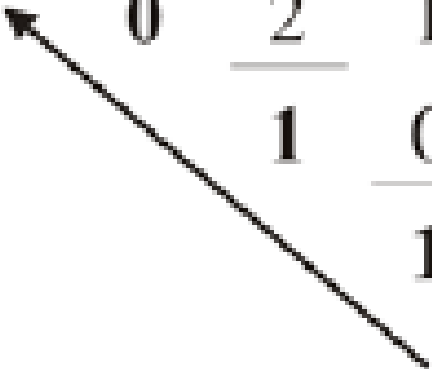
□ 11101_2

Conversione da base 10 a base 2

□ Infatti...

$$11101_2 = 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 16 + 8 + 4 + 1 = 29_{10}$$

Conversione da base 10 a base 2

$$\begin{array}{r} 12 \quad | \quad 2 \\ \hline 12 \quad 6 \quad | \quad 2 \\ \hline \mathbf{0} \quad 6 \quad 3 \quad | \quad 2 \\ \hline \quad \mathbf{0} \quad 2 \quad 1 \quad | \quad 2 \\ \hline \quad \quad \mathbf{1} \quad 0 \quad 0 \\ \hline \quad \quad \quad \mathbf{1} \end{array}$$


Esistono anche altre basi di numerazione

□ CODICE OTTALE

- cifre: 0, 1, 2, 3, 4, 5, 6, 7
- 10 (ottale) = 8 (decimale)

□ CODICE ESADECIMALE

- cifre: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
 - 10 (esadecimale) = 16 (decimale); B = 11; $2B = 2 * 16^1 + B * 16^0 = 32 + 11 = 43$
-

Aritmetica binaria

□ Somma tra numeri binari

+	0	1
0	0	1
1	1	10

Somma tra numeri binari: alcuni esempi

$$\begin{array}{r} 1 + \\ 1 = \\ \hline 1 \quad 0 \end{array}$$

$$\begin{array}{r} 1 \quad 0 \quad 1 + \\ \quad \quad 1 \quad 1 = \\ \hline 1 \quad 0 \quad 0 \quad 0 \end{array}$$

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \quad 0 + \\ \quad \quad \quad 1 \quad 0 \quad 1 = \\ \hline 1 \quad 1 \quad 1 \quad 1 \quad 1 \end{array}$$

Rappresentazione di numeri positivi e negativi

- Il bit più a sinistra rappresenta il segno del numero:

$$0 = '+' \quad 1 = '-'$$

$$1101 = -5$$

- E' indispensabile indicare il numero **N** di bit utilizzati:

- **1** bit per il segno e **N-1** bit per il modulo

- Con un byte possiamo rappresentare tutti i numeri compresi tra

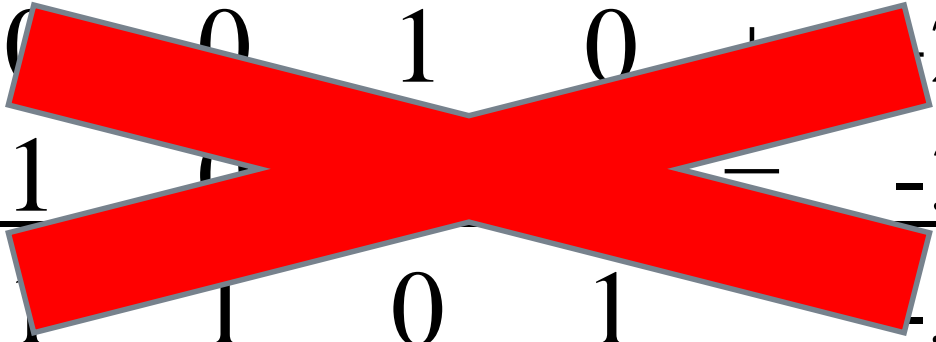
$$+127 (01111111) \text{ e } -127 (11111111)$$

- In generale con **N** bit si rappresentano i valori da

$$- 2^{N-1} - 1 \quad \text{a} \quad +2^{N-1} - 1$$

Codifica dei numeri interi negativi

- Es: Due controindicazioni:
 - 2 rappresentazioni dello 0
 - non si possono applicare le regole tradizionali per le operazioni aritmetiche:


$$\begin{array}{r} 0010 \\ + 1101 \\ \hline 10011 \end{array}$$

The image shows a binary addition of 2 (0010) and -3 (1101) resulting in 10011 (-5). A large red X is drawn over the entire calculation, indicating that traditional arithmetic rules do not apply.

Rappresentazione di numeri positivi e negativi

Complemento a 2

- Definizione: Se N sono i bit da utilizzare e x il numero da rappresentare si utilizza il valore binario pari a

$$2^N + x$$

Es. con 4 bit

$$+7 = 2^4 + 7 = 16 + 7 = 23 = \underline{\mathbf{1}}0111 = 0111$$

$$-7 = 2^4 - 7 = 16 - 7 = 9 = 1001$$

si scarta



Codifica dei numeri interi negativi: complemento a 2

- Il bit più significativo (più a sx) è per rappresentare il segno (0 per il +, 1 per il -)
- Comune rappresentazione binaria per i numeri positivi
- Per i numeri negativi: inversione dei restanti bit (0→1 e 1→0) e poi si somma 1

in alternativa:

- dati N bit, codifico in binario il numero risultato da $2^N + \text{num}$
 - (es. Con 4 bit per codificare in complemento a 2 -7 calcolo $16-7 = 9$ e codifico 9 in binario: 1001)
-

Rappresentazione in complemento a 2: esempio

- -5 con quattro bit
 - il bit di segno è 1
 - Conversione: $5_{10} = 0101_2$
 - Inversione: $0101 \rightarrow 1010$
 - Somma di 1: $1010 + 1 = 1011$
 - Verifica:
 - $+ 5 \rightarrow 0101$
 - $- 5 \rightarrow 1011$
 - $= 0 = (1)0000$
-

Conversione da complemento a 2 in decimale con segno

- se prima cifra 0 → numero positivo → conversione solita (es. 0100 → +4)

 - se prima cifra 1 → numero negativo →
 - inversione dei bit (tranne il primo)

 - conversione da binario a decimale

 - somma di 1
-

Conversione da complemento a 2 in decimale con segno: esempio

□ 1101

- tolgo il bit di segno \rightarrow 101
 - Inversione \rightarrow 010
 - Conversione in decimale $\rightarrow 010_2 = 2_{10}$
 - Somma $\rightarrow 2 + 1 = 3$
 - Segno $\rightarrow -3$
-

In generale

- Con N bit ho 2^N configurazioni possibili
 - Considerando interi positivi codifico i numeri da 0 a 2^N-1
 - Considerando interi positivi e negativi (complemento a 2) codifico i numeri:
 - positivi: da 0 a $2^{N-1}-1$
 - negativi: da -2^{N-1} a -1
-

Rappresentazione in complemento a 2

□ Con quattro bit:

0000	0	1000	-8
0001	+1	1001	-7
0010	+2	1010	-6
0011	+3	1011	-5
0100	+4	1100	-4
0101	+5	1101	-3
0110	+6	1110	-2
0111	+7	1111	-1

Applicazione del Complemento a 2: L'Addizione

- ❑ Il bit più a sinistra conserva il significato di segno.
- ❑ Il segno viene determinato automaticamente!
- ❑ Es: 15 - 5 (utilizzando 8 bit)
- ❑ Faccio la somma "normalmente"

1 1111 111 (riporto)

0000 1111 (15)

1111 1011 (-5)

=====

1 0000 1010 (10)

Regola - Se i primi due bit della riga dei riporti sono diversi, il risultato **non è valido**

NB: Si ignora il bit di overflow !



Numeri a precisione finita (1)

- I numeri a precisione finita sono quelli rappresentati con un *numero finito di cifre*.
 - Fissate le caratteristiche del numero è determinato anche l'insieme di valori rappresentabili.
- Le operazioni con i numeri a precisione finita *causano errori* quando il loro risultato non appartiene all'insieme dei valori rappresentabili:
 - *Underflow*: si verifica quando il risultato dell'operazione è minore del più piccolo valore rappresentabile
 - *Overflow*: si verifica quando il risultato dell'operazione è maggiore del più grande valore rappresentabile
 - *Non appartenenza all'insieme*: si verifica quando il risultato dell'operazione, pur non essendo troppo grande o troppo piccolo, non appartiene all'insieme dei valori rappresentabili



Numeri a precisione finita (2)

- Esempio: si considerino i numeri a tre cifre senza virgola e senza segno:
 - Non possono essere rappresentati:
 - Numeri superiori a 999
 - Numeri negativi
 - Frazioni e numeri irrazionali
 - Alcuni errori possibili in operazioni fra tali numeri:
 - $600+600 = 1200 \rightarrow \textit{Overflow}$
 - $300-600 = -300 \rightarrow \textit{Underflow}$
 - $007/002 = 3.5 \rightarrow \textit{Non appartenenza all'insieme}$

1	5	9
---	---	---



Numeri a precisione finita (3)

- L'algebra dei numeri a precisione finita è diversa da quella convenzionale, poiché alcune delle proprietà non vengono rispettate.
 - A differenza dei numeri interi, i numeri a precisione finita *non rispettano la chiusura* rispetto alle operazioni di somma, sottrazione e prodotto.
 - La *proprietà associativa* $[a + (b - c) = (a + b) - c]$ e la *proprietà distributiva* $[a \times (b - c) = a \times b - a \times c]$ non sono rispettate
- Esempi (numeri a precisione finita di 3 cifre senza virgola e senza segno):
 - Chiusura: $050 \times 050 = 2500$ (*Overflow*)
 - Prop. associativa: $(400 + 300) - 500 = 200$
 $400 + (300 - 500) = \textit{Underflow}$
 - Prop. distributiva: $50 \times (50 - 40) = 500$
 $50 \times 50 - 50 \times 40 = \textit{Overflow}$



I numeri reali e la loro rappresentazione

- Quando si parla di numeri "reali", nell'informatica, ci si riferisce sempre ad un piccolo sottoinsieme finito di numeri razionali.
- 2 possibili rappresentazioni:
 - in virgola fissa
 - In virgola mobile
- La prima rappresentazione potrebbe essere usata per applicazioni di tipo gestionale, dove l'ordine di grandezza dei numeri che compaiono non è mai troppo diverso
- La seconda invece consente di gestire numeri il cui ordine di grandezza è profondamente differente

Esempio:

Massa dell'elettrone: 0.0000000000000000000000000000000091 Kg

La massa della terra è: 597360000000000000000000 Kg

Rappresentazione di numeri frazionari in **Virgola fissa**

Un numero frazionario è rappresentato come una coppia di numeri interi: la **parte intera** e la **parte decimale**.

12,75 <12; 75>

<1100; 11> =

$$1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 1*2^{-2}$$



Numeri floating-point (1)

- Molte applicazioni richiedono il trattamento di valori razionali o reali
 - $1/3 = 0.333333\dots$ $\pi = 3.14159265\dots$
 - Non rappresentabili con un numero finito di bit
 - Per numeri molto grandi spesso interessano solo le cifre *più significative*
- Si adotta una notazione in cui la gamma dei valori esprimibili è indipendente dal numero di cifre significative. Questo sistema è detto *floating-point*.

$$n = f \times 10^e$$

frazione o mantissa *esponente*

La precisione è determinata dalla mantissa f , mentre la gamma dei valori è determinato dall'esponente e .

Esempio: Valori floating point corrispondenti alla mantissa $f=0.241$, al variare del numero delle cifre significative e dell'esponente e .

<i>c.s.</i> \ e	-3	-2	-1	0	1	2	3
1	0.0002	0.002	0.02	0.2	2	20	200
2	0.00024	0.0024	0.024	0.24	2.4	24	240
3	0.000241	0.00241	0.0241	0.241	2.41	24.1	241

Numeri in virgola mobile (**Floating point**)

Idea: $12,52 = 1252/100 = 1252 * 10^{-2}$

Un numero decimale è rappresentato come un intero moltiplicato per una opportuna potenza di 10, cioè con una coppia:

<1252; -2>

mantissa esponente

Numeri floating point

E' necessario stabilire quanti bit assegnare alla mantissa e all'esponente.

Ad esempio, con 16 bit a disposizione possiamo usarne 12 per la mantissa e 4 per l'esponente

(la mantissa e l'esponente sono di solito espressi in complemento a 2, per cui un bit corrisponde al segno della mantissa e uno a quello dell'esponente)

Numeri floating point

Con lo stesso metodo possiamo rappresentare numeri molto grandi. Ad esempio, con 8 bit:

4 bit di mantissa: $0111 = 7$

4 bit di esponente: $0111 = 7$

$$0111\ 0111 = 7 * 2^7 = 896$$

Mentre, con la notazione classica, con 8 bit rappresentiamo al massimo il n. 255

Numeri floating point

Ma allora, perchè non usare sempre la notazione floating point?

Perchè si perde in precisione

Esempio: 5 cifre (decimali) : 4 per la mantissa, 1 per l'esponente. Rappresentare

312,45

$\langle 3124; -1 \rangle = [312,4 \dots 312,5]???$



Non posso rappresentare gli infiniti numeri che si trovano qua (es. 312,41)!



Numeri floating-point (4)

- Non tutti i numeri reali appartenenti alle aree rappresentabili possono essere espressi correttamente tramite un numero floating-point.
 - Esempio: *Con numeri floating-point con tre cifre decimali con segno per la mantissa e due cifre decimali con segno per l'esponente non è possibile rappresentare $10/3=3.333333\dots$*

$$0.333 \times 10^1 < 3.\bar{3} < 0.334 \times 10^1$$

A differenza dei numeri reali, la *densità* dei numeri floating-point *non è infinita* → **Errori di arrotondamento**

- Quando il risultato v non si può esprimere nella rappresentazione numerica adottata, si utilizza il numero più vicino rappresentabile ($v_1 < v < v_2$).

Numeri floating point

Quindi: possiamo rappresentare numeri molto grandi o con molti decimali al costo di una perdita di precisione

Perchè? Perchè i computer permettono solo rappresentazioni **finite**, e così dobbiamo approssimare alcuni numeri (ad esempio gli irrazionali), ma anche **immagini e suoni**

Codifica dei caratteri alfabetici – 1

- ❑ Oltre ai numeri, molte applicazioni informatiche elaborano caratteri (simboli)
- ❑ Gli elaboratori elettronici trattano numeri
- ❑ Si codificano i caratteri e i simboli per mezzo di numeri
- ❑ Per poter scambiare dati (testi) in modo corretto, occorre definire uno standard di codifica

A	—————>	01000001
3	—————>	00110011
\$	—————>	00100100

Codifica dei caratteri alfabetici – 2

- Quando si scambiano dati, deve essere noto il tipo di codifica utilizzato
 - La codifica deve prevedere le lettere dell'alfabeto, le cifre numeriche, i simboli, la punteggiatura, i caratteri speciali per certe lingue (æ, ã, ë, è,...)
 - Lo standard di codifica più diffuso è il **codice ASCII**, per **American Standard Code for Information Interchange**
-

Codifica ASCII

- ❑ Definisce una tabella di corrispondenza fra ciascun carattere e un codice a **7 bit** (128 caratteri)
- ❑ I caratteri, in genere, sono rappresentati con **1 byte** (8 bit); i caratteri con il bit più significativo a 1 (quelli con codice dal 128 al 255) rappresentano un'estensione della codifica
- ❑ La tabella comprende sia **caratteri di controllo** (codici da 0 a 31) che **caratteri stampabili**
- ❑ I caratteri alfabetici/numerici hanno codici ordinati secondo l'ordine alfabetico/numerico

0 48	A 65	a 97
1 49	B 66	b 98
.....
8 56	Y 89	y 121
9 57	Z 90	z 122

cifre

maiuscole

minuscole

Caratteri di controllo ASCII

- I caratteri di controllo (codice da 0 a 31) hanno funzioni speciali
- Si ottengono o con tasti specifici o con una sequenza **Ctrl+carattere**

Ctrl	Dec	Hex	Code	Nota
^@	0	0	NULL	carattere nullo
^A	1	1	SOH	partenza blocco
.....
^G	7	7	BEL	beep
^H	8	8	BS	backspace
^I	9	9	HT	tabulazione orizzontale
^J	10	A	LF	line feed (cambio linea)
^K	11	B	VT	tabulazione verticale
^L	12	C	FF	form feed (alim. carta)
^M	13	D	CR	carriage return (a capo)
.....
^Z	26	1A	EOF	fine file
^[27	1 B	ESC	escape
.....
^_	31	1F	US	separatore di unità

Caratteri ASCII stampabili

Dec Hx Chr Dec Hx Chr Dec Hx Chr Dec Hx Chr Dec Hx Chr Dec Hx Chr

32	20	SPACE	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
36	24	\$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
40	28	(56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
41	29)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
43	2B	+	59	3B	;	75	4B	K	91	5B	[107	6B	k	123	7B	{
44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	
45	2D	-	61	3D	=	77	4D	M	93	5D]	109	6D	m	125	7D	}
46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Nota: il valore numerico di una cifra può essere calcolato come differenza del suo codice ASCII rispetto al codice ASCII della cifra 0 (es. '5'-'0' = 53-48 = 5)

Tabella ASCII estesa

- I codici oltre il 127 non sono compresi nello standard originario

128	Ç	144	É	160	á	176	░	193	⊥	209	⚏	225	β	241	±
129	ü	145	æ	161	í	177	▒	194	⊤	210	⚐	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	⊢	211	⚑	227	π	243	≤
131	â	147	ô	163	ú	179		196	—	212	⚒	228	Σ	244	∫
132	ä	148	ö	164	ñ	180	⊥	197	⊕	213	⚓	229	σ	245	∫
133	à	149	ò	165	Ñ	181	⊣	198	⊦	214	⚔	230	μ	246	÷
134	â	150	û	166	ª	182	⊥	199	⊧	215	⚕	231	τ	247	≈
135	ç	151	ù	167	º	183	⊤	200	⚈	216	⚖	232	Φ	248	°
136	ê	152	_	168	¿	184	⊥	201	⚉	217	⚗	233	⊕	249	.
137	ë	153	Ö	169	_	185	⊥	202	⚊	218	⚘	234	Ω	250	.
138	è	154	Û	170	¬	186	⊥	203	⚏	219	■	235	δ	251	√
139	ï	156	£	171	½	187	⊥	204	⚏	220	■	236	∞	252	_
140	î	157	¥	172	¼	188	⊥	205	=	221	■	237	φ	253	²
141	ï	158	_	173	¡	189	⊥	206	⚏	222	■	238	ε	254	■
142	Ä	159	f	174	«	190	⊥	207	±	223	■	239	∩	255	
143	Å	192	ℓ	175	»	191	⊥	208	⚈	224	α	240	≡		