Programma del corso

- *Introduzione agli algoritmi*
- ***Rappresentazione delle Informazioni***
- *Architettura del calcolatore*
- *Reti di Calcolatori (Reti Locali, Internet)*
- *Elementi di Programmazione*

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

Rappresentazione delle informazioni

Idea di fondo

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

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 $2^N >= M$

Esempio

Per rappresentare **57** informazioni diverse:

5 bit non bastano, poiché $2^5 = 32 < 57$

6 bit invece bastano: $2^6 = 64 > 57$

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

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

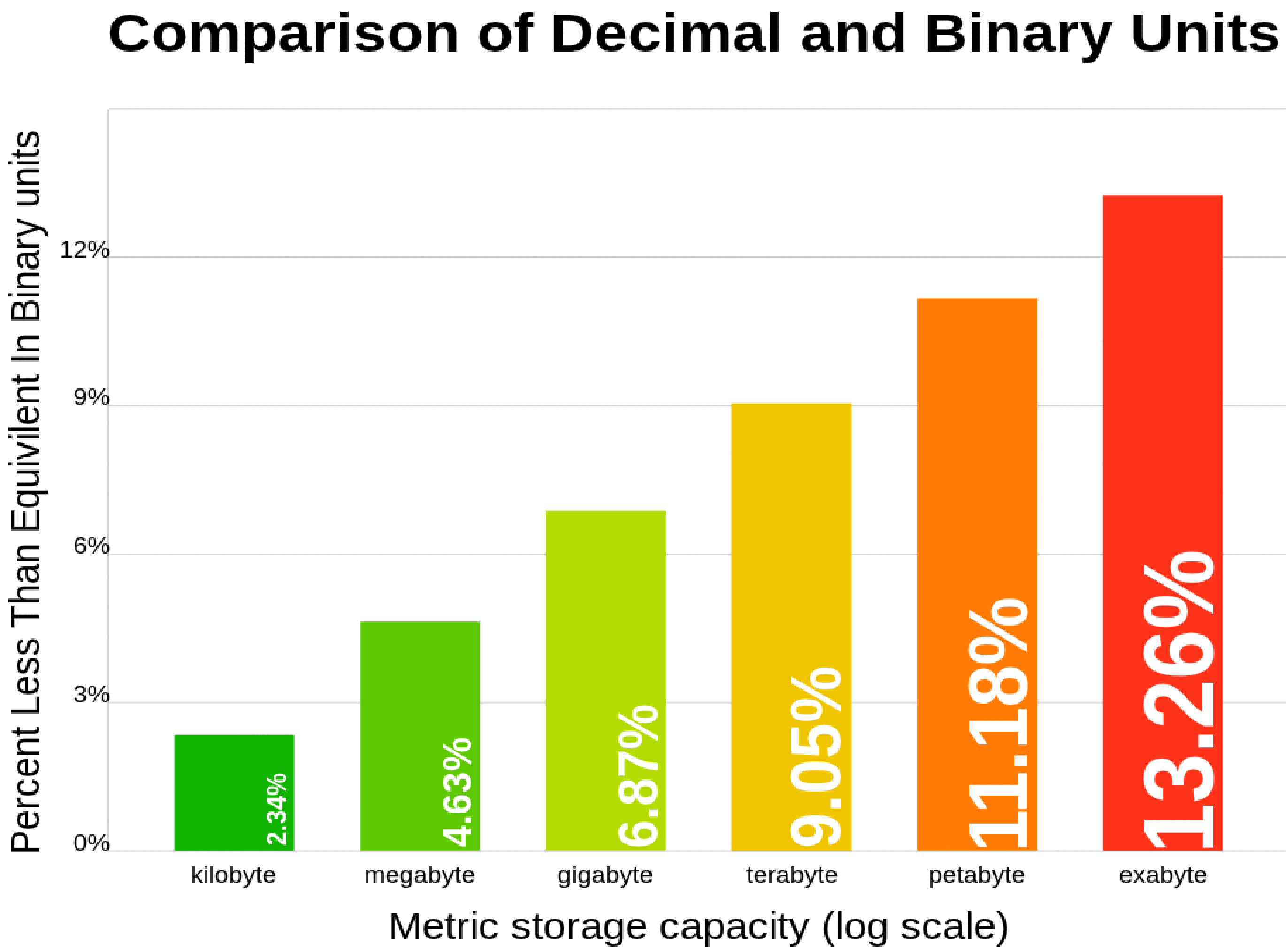
Altre unità di misura

- 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 SI: 1 Kilobyte = 1000 byte
- Sistema IEC: 1 “Kilo”byte (detto *Kibibyte* = 1024 byte)

- Più precisamente (sistema IEC)
  - 1 KiB = 1024 x 1 byte = $2^{10} \sim 10^3$ byte
  - 1 MiB = 1024 x 1 KiB = $2^{20} \sim 10^6$ byte
  - 1 GiB = 1024 x 1 MiB =$2^{30} \sim 10^9$ byte

- Normalmente il sistema IEC è usato come unità di misura per la capacità della memoria di un elaboratore.
- Normalmente il sistema SI è usato come unità di misura per le capacità degli hard disk (purtroppo non da tanti sistemi operativi).

Discrepanza SI/IEC



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: numero rappresentato in notazione decimale

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

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

$$2000 + 300 + 0 + 4 = 2304_{10}$$ (**numero**)

Nota: la notazione del numero e il numerale qui coincidono, perché il sistema decimale e quello adottato come sistema di riferimento.

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)
  - $2^3=8$; $2^4=16$; $2^5=32$; $2^6=64$; $2^7=128$; $2^8=256$; $2^9=512$; $2^{10} = 1024$; $2^{11}=2048$, $2^{12}=4096$;...

Esempio di numero rappresentato in notazione binaria

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

$$10100101_2 =$$

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

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

165_{10} (**numero**)

Il numero più grande rappresentato con N cifre

- Sist. Decimale = $99...99_{10}$ = $(10^N - 1)_{10}$
- Sist. Binario= $11..11_2$ = $(2^N - 1)_{10}$
- **Esempio**: 11111111_2 (8 bit binari) = $(2^8 -1)_{10}$ = 255_{10}.

  Per rappresentare il numero 256_{10} ci vuole un bit in più:

  100000000_2 = $(1*2^8)_{10}$ = 256_{10}.

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: $(2^{16} - 1)_{10} = 65\ 535_{10}$
- con 32 bit: $(2^{32} - 1)_{10} = 4\ 294\ 967\ 295_{10}$
- con 64 bit: $(2^{64} - 1)_{10}$ = circa $(1{,}84 * 10^{19})_{10}$

Conversione da base 2 a base 10

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

Esempio:

$$10100_2 =$$

$$(1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0)_{10} =$$

$$(16 + 4)_{10} = 20_{10}$$

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

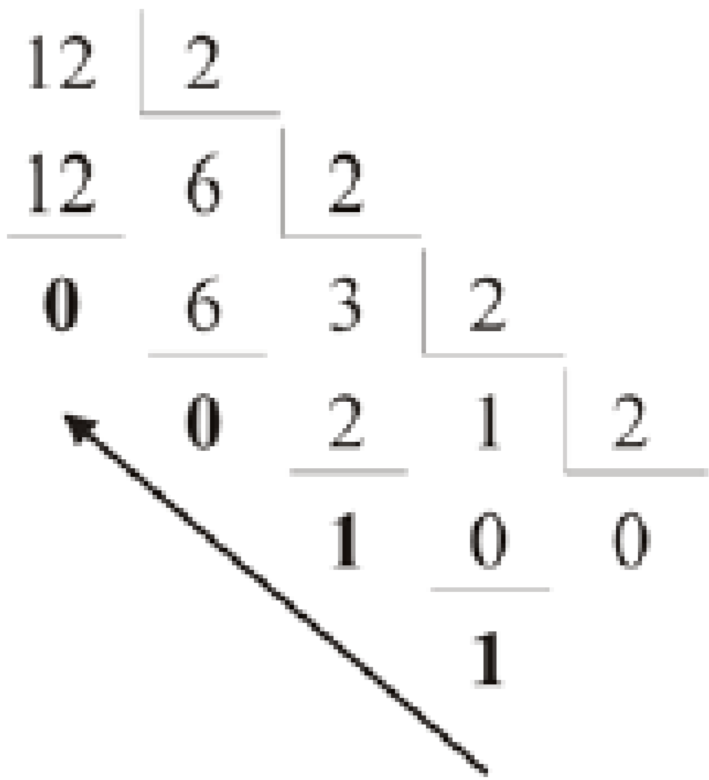
- Dividere il numero per 2 ripetutamente finche` il risultato non e` 0
- Scrivere i resti in ordine inverso.

Esempio: conversione del numero 12_{10}

Divisioni:	12/2=6;	6/2=3;	3/2=1;	1/2=0
Resti:	0	0	1	1

$$12_{10} = 1100_2$$

Conversione da base 10 a base 2



Esistono anche altre basi di numerazione

- **CODICE OTTALE (base 8)**
  - cifre: 0, 1, 2, 3, 4, 5, 6, 7
  - $10_8 = (1*8^1+0*8^0) = 8_{10}$; $11_8 = 9_{10}$; $21_8 = 17_{10}$
- **CODICE ESADECIMALE (base 16)**
  - cifre: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - $10_{16} = 16_{10}$; $B_{16} = 11_{10}$;

    $2B_{16} = (2*16^1+B*16^0)_{10} = (32+11)_{10} = 43_{10}$

Rappresentazione di numeri positivi e negativi

Fino ad adesso abbiamo rappresentato solo numeri positivi! Per rappresentare anche numeri negativi, ci sono metodi vari:

- Segno e grandezza
- Complemento a 2

Rappresentazione di numeri positivi e negativi: Segno e grandezza

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

  $0 \rightarrow$ '+' $1 \rightarrow$ '-'

  $$1101_{SG} = -5_{10}$$

- E' indispensabile indicare il numero **N** di bit utilizzati:
  - **1** bit per il segno e **N-1** bit per il valore assoluto
- Con un byte possiamo rappresentare tutti i numeri compresi tra

  $+127_{10}$ (01111111_2) e -127_{10} (11111111_2)

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

  $\mathbf{(-2^{N-1} - 1)_{10}}$ **a** $\mathbf{(+2^{N-1} - 1)_{10}}$

Segno e grandezza: Vantaggi e svantaggi

Vantaggi:

- Conversione semplice

Svantaggi:

- Due valori per 0
- Addizione non “automatica”
- Numeri negativi “più grandi” di numeri positivi

Rappresentazione di numeri positivi e negativi: Complemento a 2

- Se N sono i bit a disposizione e x il numero (positivo o negativo, tra $-(2^{N-1})$ e $+(2^{N-1}-1)$) da rappresentare, si utilizza il valore binario pari a

$$2^N + x$$

scartando un eventuale N+1-esimo bit.

Esempio con 4 bit

da scartare

$$+7_{10} \Rightarrow (2^4 + 7)_{10} = (16 + 7)_{10} = 23_{10} = \underline{\mathbf{1}}0111_2 \Rightarrow 0111$$

$$-7_{10} \Rightarrow (2^4 - 7)_{10} = (16 - 7)_{10} = 9_{10} = 1001_2 \Rightarrow 1001$$

Rappresentazione di numeri positivi e negativi: Complemento a 2

- In alternativa, per i numeri negativi si eseguono i seguenti passi:
  - Si rappresenta in binario il corrispondente numero positivo
  - Si invertono tutti i bit
  - Si aggiunge 1

Esempio con 4 bit:

Inversione dei bit

$$-7 \Rightarrow +7_{10} = 0111_2 \Rightarrow (1000+1)_2 = 1001_2$$

Rappresentazione di numeri positivi e negativi: Complemento a 2

- Ancora un'altro metodo:
  - Si rappresenta in binario il corrispondente numero positivo
  - Da destra, si copia tutte le 0 e il primo 1
  - Dopo il primo 1 si invertono tutti i bit

Esempio con 4 bit:

$-6 \Rightarrow +6_{10} = 0110_2 \Rightarrow 0110 \Rightarrow 1010$

Dal Complemento a 2 al Decimale

Moltiplicare ogni bit per il suo peso posizionale, il bit più a sinistra anche con -1, sommare il tutto:

Esempi con complemento a due a 4 Bit:

$$0111_{C2} = (0*(-1)*2^3 + 1*2^2 + 1*2^1 + 1*2^0)_{10} = (4 + 2 + 1)_{10} = +7_{10}$$

$$1001_{C2} = (1*(-1)*2^3 + 0*2^2 + 0*2^1 + 1*2^0)_{10} = (-8 + 1)_{10} = -7_{10}$$

Addizione di Numeri al Complemento di 2

- Tramite l'addizione normale a base di 2!
- Con N bit, eventuali N+1-esimi bit nel risultati sono scartati
- Il segno viene determinato “automaticamente”.
- Esempio: 15 + –5 (utilizzando 8 bit)

1 1 1 1 1   1 1 1   (riporto)

0000 1111 ⇨ 15_{10}

1111 1011 ⇨ -5_{10}

===========

<u>1</u> 0000 1010 ⇨ 0000 1010 ⇨ 10_{10}

8+1-esimo bit viene scartato

Complemento a due: Vantaggi e svantaggi

Vantaggi:

- Addizione “automatica”
- Un solo valore per 0
- Ordine dei numeri mantenuto

Svantaggi:

- Conversione leggermente più complicata

Rappresentazione di numeri frazionari: Virgola fissa

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

12,75 scritto come una coppia: <12;0,75>

<1100; 11> ⇨

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

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 (binari)

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 (binari)

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

<0111;0111>

4 bit di mantissa: $0111_2 = 7_{10}$

4 bit di esponente: $0111_2 = 7_{10}$

<0111;0111> ⇨ $(7 * 2^7)_{10} = 896_{10}$

Mentre, con la notazione classica, con 8 bit rappresentiamo al massimo il numero 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

<3124; -1> = [312,4 .. 312,5]???

Numeri floating point

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

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

La Codifica dei Caratteri

AB...ab...&%$...

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 à, è, €, … bastano 8 bit nelle codifiche accettate universalmente chiamata **ASCII esteso**
- Per manipolare un numero maggiore di simboli si utilizzano codifiche di **UNICODE**

Codifica ASCII

- La codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) 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)
- **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 caratteri e **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)