Codifica binaria dell' informazione

Codifica binaria dell'informazione

- Tutte le informazioni vanno tradotte in bit (organizzati poi in *Byte*):
  - Numeri naturali
  - Numeri interi (con segno)
  - Numeri frazionari
  - Numeri reali
  - Caratteri
  - Immagini
- Nell'interazione con il calcolatore sia la codifica in binario che la decodifica in formato leggibile per l'operatore umano avvengono in modo trasparente all'utente

Numeri naturali: Sistemi di numerazione

- Un sistema di numerazione è composto da:
  - Insieme finito di simboli (o cifre)
  - Regole che permettono di rappresentare i numeri
- Classificazione
  - Sistemi additivi (Es. sistema romano, con alcuni accorgimenti …):
    - Ogni cifra assume un valore prefissato
    - Il numero si ottiene addizionando le cifre che lo compongono (…)
    - Impossibilità di rappresentare numeri molto grandi e difficoltà di esecuzione delle operazioni matematiche
  - Sistemi posizionali (Es. sistema decimale):
    - Le cifre hanno peso diverso a seconda della posizione che occupano
    - Un numero di n cifre è rappresentato in base *p* dalla sequenza:

      $a_{n-1}, a_{n-2},\dots, a_0$

    - Compattezza di rappresentazione anche per numeri molto grandi e facilità di esecuzione delle operazioni

Sistemi pos.:rappresentazione in base p

Numero naturale composto da *n* cifre, in base *p*:

- **Rappresentazione:**

  Il numero $a_{n-1}a_{n-2}\ldots a_0$ in base p rappresenta il valore:

$$a_{n-1}p^{n-1}+a_{n-2}p^{n-2}+\ldots+a_1p^1+a_0p^0 = \sum_{i=0}^{n-1} a_i p^i$$

- **Spazio di Rappresentazione (range)**:

  Con n cifre, in base p si possono rappresentare tutti i numeri nell'intervallo $[0 \ , \ p^n - 1]$

ISII

Sistema decimale: rappresentazione in base 10

Sistema posizionale

- Esempio: 123 = **1**00 +**2**0 +**3**

Base: p = 10

Insieme di simboli: $a_i \in \{0,1,2,3,4,5,6,7,8,9\}$

Numero naturale *N* di *n* cifre:

- Rappresentazione:
  - $N_{10} = a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2} + \ldots + a_0 \cdot 10^0$
  - Esempio, con n=3:
    $587_{10} = 5 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$
- Spazio di rappresentazione: intervallo discreto $[0, 10^n-1]$
  - Con n=3, range: $[0, 10^3-1]$ ossia [0, 999]

Sistema binario:rappresentazione in base due

Sistema posizionale

Base binaria: $p=2$

Insieme di simboli: $a_i \in \{0, 1\}$

- Simboli chiamati *bit* (*binary digit*)
- Otto bit chiamati *Byte (abbreviato con B)*

Numero naturale *N* di n cifre:

- Rappresentazione:
  - $N_2 = a_{n-1} \cdot 2^{n-1} + \ldots + a_0 \cdot 2^0$

Esempio, con n=*5*: $11011_2 = (1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = 27_{10}$

- Spazio di rappresentazione:
  - intervallo discreto $[0 \ , \ 2^m - 1]$

Esempio con n=*8*: $[00000000_2 \ , \ 11111111_2]$, ovvero: $[0_{10} \ , \ 256-1=255_{10}]$

Sistema binario: unità di misura

- Kilobyte (KB) = 2^{10} byte = 1024 byte
- Megabyte (MB) = 2^{20} byte = 1048576 byte
- Gigabyte (GB) = 2^{30} byte = 1073741824 byte
- Terabyte (TB) = 2^{40} byte = 1099511627776 byte

Unità di misura (da wikipedia)

Prefissi del Sistema Internazionale

10^n	Prefisso	Simbolo	Nome	Equivalente decimale
10^{24}	yotta	Y	Quadrilione	1 000 000 000 000 000 000 000 000
10^{21}	zetta	Z	Triliardo	1 000 000 000 000 000 000 000
10^{18}	exa	E	Trilione	1 000 000 000 000 000 000
10^{15}	peta	P	Biliardo	1 000 000 000 000 000
10^{12}	tera	T	Bilione	1 000 000 000 000
10^9	giga	G	Miliardo	1 000 000 000
10^6	mega	M	Milione	1 000 000
10^3	kilo o chilo	k	Mille	1 000
10^2	etto	h	Cento	100
10	deca	da	Dieci	10
10^{-1}	deci	d	Decimo	0,1
10^{-2}	centi	c	Centesimo	0,01
10^{-3}	milli	m	Millesimo	0,001
10^{-6}	micro	µ	Milionesimo	0,000 001
10^{-9}	nano	n	Miliardesimo	0,000 000 001
10^{-12}	pico	p	Bilionesimo	0,000 000 000 001
10^{-15}	femto	f	Biliardesimo	0,000 000 000 000 001
10^{-18}	atto	a	Trilionesimo	0,000 000 000 000 000 001
10^{-21}	zepto	z	Triliardesimo	0,000 000 000 000 000 000 001
10^{-24}	yocto	y	Quadrilionesimo	0,000 000 000 000 000 000 000 001

Basi "significative": ottale ed esadecimale

- **Rappresentazione in base 8:**
  - Base ottale: *p=8*;
  - Insieme di simboli $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
  - Numero N di n cifre:
  - Rappresentazione: $N_8 = (a_{n-1} \cdot 8^{n-1} + \ldots + a_0 \cdot 8^0)_{10}$

Es. $234_8 = (2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0)_{10} = 156_{10}$

  - Spazio di rappresentazione: $[0, 8^n-1]$

- **Rappresentazione in base 16:**
  - Base esadecimale: *p=16*;
  - Insieme di simboli $a_i \in \{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$
  - Notare: "10" al posto di "A" … "15" al posto di "F"
  - Numero N di n cifre:
  - Rappresentazione: $N_{16} = (a_{n-1} \cdot 16^{n-1} + \ldots + a_0 \cdot 16^0)_{10}$

Es. $B7F_{16} = (11 \cdot 16^2 + 7 \cdot 16^1 + 15 \cdot 16^0)_{10} = 2943_{10}$

  - Spazio di rappresentazione: $[0, 16^n-1]$

Conversioni: esempi

- Base 2

$1\,0\,1\,0_{\,2} = 2^3 + 2^1 = 10_{\,10}$

$1\,1\,0\,0\,1\,0\,0_{\,2} = 2^6 + 2^5 + 2^2 = 100_{\,10}$

$1\,1\,1\,1\,1\,0\,1\,0\,0\,0_{\,2} = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 = 1000_{\,10}$

- Base 8

$1\,2_{\,8} = 8 + 2 = 10_{10}$

$1\,4\,4_{\,8} = 8^2 + 4\bullet 8^1 + 4 = 100_{10}$

$1\,7\,5\,0_{\,8} = 8^3 + 7\bullet 8^2 + 5\bullet 8^1 = 512 + 448 + 40 = 1000_{\,10}$

Conversioni: esempi

Base 16

$A_{16} = 10_{10}$

$6\,4_{16} = 6 \bullet 16 + 4 = 100_{10}$

$3\,E\,8_{16} = 3 \bullet 16^2 + 14 \bullet 16 + 8 = 768 + 224 + 8 = 1000_{10}$

$F_{16} = 15_{10}$

$5\,E_{16} = 5 \bullet 16 + 14 = 94_{10}$

$7\,1\,B_{16} = 7 \bullet 16^2 + 1 \bullet 16 + 11 = 1792 + 16 + 11 = 1819_{10}$

Conversioni: altri esempi

$110100_2 =$

$1\bullet 2^5 + 1\bullet 2^4 + 0\bullet 2^3 + 1\bullet 2^2 + 0\bullet 2 + 0\bullet 1 = 32 + 16 + 4 = 52_{10}$

$3216_8 =$

$3\bullet 8^3 + 2\bullet 8^2 + 1\bullet 8 + 6 = 1536 + 128 + 8 + 6 = 1678_{10}$

$AB9E_{16} =$

$10\bullet 16^3 + 11\bullet 16^2 + 9\bullet 16 + 15 = 40960 + 2816 + 144 + 15 = 43935_{10}$

Conversioni di base: da 10 a 2

- Per convertire da base p a base 10:

$$N_p=a_{n-1}p^{n-1}+a_{n-2}p^{n-2}+\ldots+a_1p^1+a_0p^0 = \sum_{i=0}^{n-1} a_i p^i$$

Esempio: $11011_2 = (1\cdot2^4+1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0)_{10} = 27_{10}$

- Per convertire da base dieci a base due:
  - Metodo delle divisioni successive: esempio

331:2	= 165	con resto di 1
165:2	= 82	con resto di 1
82:2	= 41	con resto di 0
41:2	= 20	con resto di 1
20:2	= 10	con resto di 0
10:2	= 5	con resto di 0
5:2	= 2	con resto di 1
2:2	= 1	con resto di 0
1:2	= 0	con resto di 1

$331_{10}= 101001011_2$

Conversioni di base: da 10 a 2

- Più schematicamente:

Quoziente	Resto
331	1
165	1
82	0
41	1
20	0
10	0
5	1
2	0
1	1

$331_{10}=101001011_2$

Leggere la sequenza dei resti dal basso verso l'alto…

Conversione di base: da 10 a 2

1258	0
629	1
314	0
157	1
78	0
39	1
19	1
9	1
4	0
2	0
1	1
0	

$1258_{10} = 10011101010_2$

Conversione di base: da 10 a 8

1258	2
157	5
19	3
2	2
0	

$1258_{10} = 2352_{8}$

Conversione di base: da 10 a 16

1258	10
78	14
4	4
0	

$1258_{10} = 4EA_{16}$

Conversioni di base

- Le basi ottale ed esadecimale sono di interesse informatico per la facilità di conversione, (“per parti”):
  - Da base 2 a base 8: si converte a gruppi di tre bit, traducendo ciascuna tripla nella corrispondente cifra ottale

$$\underbrace{0\ 0\ 1}_{1}\ \underbrace{0\ 1\ 0}_{2}\ \underbrace{1\ 1\ 0}_{6}\ \underbrace{1\ 1\ 1}_{7}\ {}_2 = 1267_8$$

  - Da base 2 a base 16: si converte a gruppi di quattro bit, traducendo ciascuna quadrupla nella corrispondente cifra esadecimale

$$\underbrace{0\ 0\ 1\ 0}_{2}\ \underbrace{1\ 0\ 1\ 1}_{B}\ \underbrace{0\ 1\ 1\ 1}_{7}\ {}_2 = 2B7_{16}$$

- Le basi ottale ed esadecimale consentono una consistente sintesi rispetto al formato binario

La somma in base p

- Viene effettuata rispettando le regole che conoscete in base 10.
- Quando si genera un risultato maggiore o uguale alla base, mancando il simbolo per poterlo rappresentare, si riconduce ad un valore minore della base (sottraendo il valore della base stessa) generando un riporto (carry)

- Seguono le tabelle relative all'aritmetica in base 2:

+	0	1
0	0	1
1	1	10

Addendi		Riporto	Risultato
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

La somma in base n (naturali)

- Viene eseguita incolonnando i numeri e sommando tra loro i bit incolonnati, partendo dai meno significativi, in ordine di peso crescente.
- Per la somma di due numeri positivi di lunghezza K possono essere necessari K+1 posti. Se sono disponibili solo K cifre si genera un errore di **overflow** (o trabocco).

Primo esempio: 11011_2+00110_2

11011 +	(27)
00110 =	(6)
1 00001	(33)

Esempio di somma in base 2 con carry

- Esempio:

```
  1   ← riporto
0101 +        (510)
1001 =        (910)
------
1110          (1410)
```

```
          111   ← riporti
           1111 +     (15_10)
           1010 =     (10_10)
          -------
carry →  11001
```

Risulta 25_{10} se uso 5 bit; ma 9_{10} se considero solo 4 bit: errato!

Sottrazione tra numeri binari interi positivi

Regole base:

- 0-0=0
- 0-1=1 con prestito di 1
- 1-0=1
- 1-1=0

Esempio:

```
 01000111      ← prestiti
  10101110 -
  01000111 =
  -------------------
  01100111
```

Moltiplicazione tra numeri binari interi positivi

Valgono le regole che già conosci, è però più semplice effettuare la moltiplicazione in quanto le cifre sono solo 0 ed 1…

```
     110 x
     101 =
    ------
     110
    000
   110
  000
 000
 -------
 0011110
```

Gli zeri iniziali possono essere eliminati

Moltiplicazione: potenze di 2

Nel caso di moltiplicazione per potenza k-esima di 2 il risultato è uno shift a sinistra di k posizioni:

```
    110 x
     10 =
   ------
   000
  110
-------
  1100
```

```
     110 x
     100 =
  --------
  00000
 00000
  110
---------
  11000
```

Divisione tra Numeri Binari

```
10011011 | 1001
1001     |--------
00001011 | 10001
    1001 |
    0010 |
```

quoziente

resto

Verifichiamo la correttezza del calcolo:

```
      1001*
     10001=
     ------
      1001
     0000
    0000
   0000
  1001
  --------
  10011001+
      0010=  (resto)
  --------
  10011011
```

Numeri interi

- Includono anche i numeri negativi
- Rappresentati tramite il segno ed il valore del numero
- Codifica binaria secondo uno delle due modalità seguenti

  - Rappresentazione in **modulo e segno**
  - Rappresentazione in **complemento a due**

Modulo e segno

- In un numero di n bit il primo bit è utilizzato per memorizzare il segno:
  - "1" numero negativo
  - "0" numero positivo
- Spazio di rappresentazione: tra $-(2^{n-1}-1)$ e $+(2^{n-1}-1)$
- "Stranezza" dello zero *positivo* e *negativo (doppia rappresentazione)*

Esempio *n=3*

Intero, base 10	Intero, base due, modulo e segno
–3	111
–2	110
–1	101
–0	100
+0	000
+1	001
+2	010
+3	011

Complemento a due (C2)

- Il MSB ha peso negativo, così tutti i numeri negativi cominciano con il bit più significativo posto a “1”, mentre tutti i positivi e lo zero iniziano con uno “0”
- Usando n bit: $(-N)_{C2} = (2^n - N_{10})_2$ vedi tabella sottostante
- Spazio di rappresentazione: intervallo $[-2^{n-1}\ ,\ 2^{n-1} - 1]$
  - Asimmetria tra negativi e positivi
  - Esempio (con n=*8*): [-128, +127], perché $-2^7 = -128$ e $2^7 - 1 = +127$
- Occorre sempre concordare il numero di bit usati per rappresentare il numero!!!

Esempio *n=3*

$(-N)_{C2} = (2^3 - N_{10})_2$

Num. intero base 10	Trasformazione	Num. intero, base 2,C2, *n=3*
-4	8 - 4 = 4	$4_{10} = 100$
-3	8 - 3 = 5	$5_{10} = 101$
-2	8 - 2 = 6	$6_{10} = 110$
-1	8 - 1 = 7	$7_{10} = 111$
0	nessuna	$0_{10} = 000$
1	nessuna	$1_{10} = 001$
2	nessuna	$2_{10} = 010$
3	nessuna	$3_{10} = 011$

Complemento a due (C2)

Metodo per ottenere -NC2 avendo già la configurazione binaria di N

Ricopiare i bit del modulo N da destra limitatamente a:

- Tutti gli zeri consecutivi a partire da dx
- Ricopiare anche il primo 1 incontrato a partire da dx
- Complementare tutti gli altri bit (0 diventa 1 e viceversa)

Complemento a due (C2)

- Metodo alternativo per ottenere $-N_{C2}$
  - Complementare i bit della rappresentazione binaria del modulo N (cambiare gli 1 in 0 e viceversa)
  - Sommare 1 al risultato ottenuto

Esempio: $-N = -3$ $N=3_{10}=011_2$

complemento ad 1 100

complemento a 2 101

Somma e sottrazione in C2

- Somma: come per i naturali
- Sottrazione: $N_1 - N_2 = N_1 + (-N_2)_{C2}$
- Carry:
  - Il carry finale non viene considerato!
- Overflow:
  - Se, sommando due interi di *n* bit dotati di segno concorde, ottengo un risultato di segno discorde (sempre considerando *n* bit), allora si ha un *overflow* (il risultato non è codificabile su *n* bit) e l'operazione è errata
  - L'overflow non può verificarsi se gli operandi sono di segno discorde

Somma e sottrazione in C2

Esempi: n=7 spazio di rappresentazione [-64, +63]

+5	0000101
+8	0001000
+13	0001101

+5	0000101
-8	1111000
-3	1111101

-5	1111011
+8	0001000
+3	(1)0000011

RIPORTO

-64	1000000
-8	1111000
-72	[1](1)0111000

OVERFLOW   RIPORTO

Numeri frazionari

Rappresentazione:

- Relativa alla parte frazionaria
- Ottenuta tramite la formula

$$N_p = a_{-1}p^{-1} + a_{-2}p^{-2} + \ldots + a_{-n}p^{-n} = \sum_{i=-n}^{-1} a_i p^i$$

Spazio di rappresentazione:

- Per un numero di *n* cifre in base *p*, posso rappresentare numeri nell'intervallo continuo: $[0\ ,\ 1-p^{-n}]$, ad esempio con 8 cifre dopo la virgola il range sarà $[0, 1-2^{-8}]$

Errore di approssimazione:

- minore di p^{-n}

Esempi con $n=3$:

- base 10: Rappresentazione: $(0,587)_{10} = (5\cdot10^{-1}+8\cdot10^{-2}+7\cdot10^{-3})$

  Spazio di rapp.: $[0, 1-10^{-3}] = [0, 0.999]$

  Errore : minore di 0.001

- base 2: Rappresentazione: $(0,101)_2 = (1\cdot2^{-1}+0\cdot2^{-2}+1\cdot2^{-3})10 = (0,625)_{10}$

  Spazio di rapp.: $[0, 1-2^{-3}]$

  Errore : minore di 2^{-3}

Conversioni di base: parte frazionaria

Per convertire da base p a base 10, vale lo stesso discorso fatto per i decimali riguardo le cifre dopo la virgola. Ad esempio:

$10101.1101_2 =$

$1*2^4 + 1*2^2 + 1*2^0 + 1*2^{-1} + 1*2^{-2} + 1*2^{-4} = 21.8125_{10}$

Conversioni di base: parte frazionaria

- Vedremo solo da base 10 a base 2:
  - Si moltiplica progressivamente per 2 la parte frazionaria
  - Si prendono le parti intere di ciascun prodotto dalla più alla meno significativa, con numero di bit proporzionale all'accuratezza
  - Esempio: 0.587_{10}

0.587*2= 1.174	parte intera 1	parte frazionaria 0.174
0.174*2= 0.348	parte intera 0	parte frazionaria 0.348
0.348*2= 0.696	parte intera 0	parte frazionaria 0.696
0.696*2= 1.392	parte intera 1	parte frazionaria 0.392
0.392*2= 0.784	parte intera 0	parte frazionaria 0.784
0.784*2= 1.568	parte intera 1	parte frazionaria 0.568

…..

Risultato : 0.1001 con quattro cifre e approssimazione accurate entro il limite 2^{-4}

0.100101 con sei cifre e approssimazione accurate entro il limite 2^{-6}

Numeri reali

- Modalità di rappresentazione alternative:
  - virgola fissa (Fixed Point)
  - virgola mobile (Floating Point)

Rappresentazione in virgola fissa

Data una sequenza di bit, si assume che la posizione della virgola sia fissata in un preciso punto all'interno della sequenza:

Parte intera	.	Parte frazionaria
M bit		N bit

Virgola fissa

- Uso di *m* bit per parte intera e *n* bit per parte frazionaria con *n* ed *m* fissi
  - Esempio (*m=8*, *n=6,* tot. 14 bit): $-123{,}21_{10}$
    $-\ 123_{10} = 10000101_2$
    $-\ 0{,}21_{10} \approx 001101_2$
    $-\ 123{,}21_{10} \approx 10000101{,}001101_2$

- *m* e *n* scelti in base alla precisione che si vuole tenere

Virgola mobile (floating point)

- Il numero è espresso come: $r = m \cdot b^n$
  - m e n sono in base p
  - m: mantissa (numero frazionario con segno)
  - b: base della notazione esponenziale (numero naturale)
  - n: caratteristica (numero intero)
  - Esempio (p=10, b=10):

    $-331{,}6875 = -0{,}3316875 \cdot 10^3$ $\quad m = -0{,}3316875$

    $n = 3$

Virgola mobile (floating point)

- Quando la mantissa comincia con una cifra diversa da zero, il numero in virgola mobile si dice *normalizzato*
  Es. $-0{,}3316875 \cdot 10^3$ è normalizzato perché la mantissa è "3316875"
- La normalizzazione permette di avere, a parità di cifre usate per la mantissa, una maggiore precisione.

Es. Uso 5 cifre per la mantissa:
$+45{,}6768 \approx +0{,}\mathbf{45676} \cdot 10^2 \approx +0{,}\mathbf{00456} \cdot 10^4$

Ho perso 0,0008

Ho perso 0,0768!!

Standard IEEE 754

In binario si usa una forma normalizzata (standard IEEE 754) su 32/64 bit suddivisi nel seguente modo:

Segno **Caratteristica** **Modulo della m.**

S	Esponente	Mantissa

Singola precisione (SP)	1 bit	8 bit	23 bit
Doppia precisione (DP)	1 bit	11 bit	52 bit

Standard IEEE 754

S	Esponente	Mantissa (modulo)

E’ sempre normalizzata nella forma 1.mmmm; l’1 prima della virgola viene sottinteso (hidden bit)

E’ rappresentato come numero senza segno ad 8 bit in eccesso (leggi “+”) 127, per non dover gestire il segno dell’esponente.

0 = positivo
1 = negativo

Standard IEEE 754

La rappresentazione floating point IEEE 754 è quindi nella forma:

Numero binario $= \boxed{\pm 1} . \boxed{mmmmm} * 2^{\boxed{eeee}}$

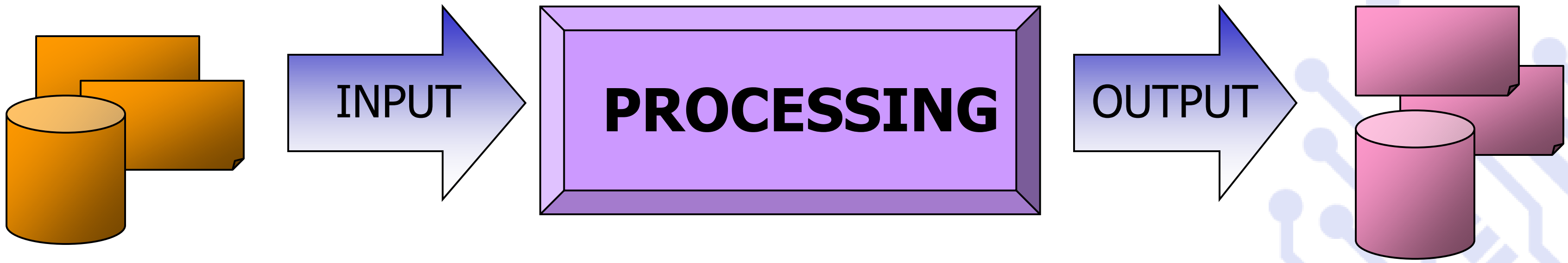
0/1	Esponente	Mantissa (modulo)

Glossario di ripasso

Seguono alcune definizioni, che verranno approfondite nel seguito del corso…

AUTOMA

L'automa è un sistema che (di regola imitando il comportamento umano), è in grado di ricevere informazioni dall'esterno (input), reagire alle stesse elaborandole (processing), e inviare le informazioni di nuovo all'esterno (output).



Il computer è un tipo di automa composto da **componenti elettronici**

Informazione analogica

Si definisce ***analogico*** un procedimento che rappresenta un fenomeno con continuità, per esempio:

•un orologio classico che con il moto regolare della lancetta segna il trascorrere del tempo in modo continuo

•oppure, un termometro che rappresenta la temperatura lancetta

ISII

Informazione digitale

E’ ***digitale*** un procedimento che rappresenta lo stesso fenomeno traducendolo **in cifre** (dall'inglese *digit*) e quindi in modo discreto, come per esempio avviene in un orologio a cristalli liquidi numerico, nel quale la stessa durata temporale viene misurata da una successione di scatti, oppure un termometro digitale

ISII

Analogico e digitale

Contrariamente a quanto si potrebbe credere la registrazione digitale, pur procedendo "a salti", può essere più precisa di quella analogica, in quanto meno soggetta ad interferenze e disturbi. Occorre però che il numero di valori utilizzati sia elevato, in modo da cogliere ogni *sfumatura* che possa essere significativa per il destinatario dell'informazione.

E' necessario, dunque, che l'errore introdotto dal procedimento di digitalizzazione non sia troppo elevato.

Frequenza di una CPU

- Il tempo in un computer non è continuo, ma “discreto”
- Ogni volta che “scatta” l’orologio interno la CPU esegue un’operazione
- La “frequenza” dell’orologio viene misurata in “Hertz” (cicli al secondo)
- Quindi “Pentium 4 – 2.0 GHz” vuol dire “due miliardi di scatti di orologio al secondo” – uno scatto ogni mezzo nanosecondo
- Quindi, due miliardi di operazioni...?

CPU

- .... non esattamente!
- Due miliardi di “microoperazioni” o “microistruzioni”
- Esempio : calcola c = a + b (somma di due variabili in un programma)
    - Leggi dalla memoria il valore di a
    - Leggi dalla memoria il valore di b
    - Passa a e b all’ALU e dille di calcolare a + b (bit a bit!)
    - Prendi il valore di a+b dall’ALU e salvalo in memoria
- Quindi, la “velocità netta” di un computer non dipende solo dalla frequenza del processore, ma anche da:
    - Tempi di accesso alla memoria
    - “Ottimizzazione” dei circuiti che eseguono calcoli
    - “Sovrapposizione” di operazioni (PIPELINING)
    - Soluzioni “multi-core” (tipo “dual core” Pentium di Intel)