Rappresentazione dell'informazione

Architettura dei Calcolatori

Prof. Andrea Marongiu
andrea.marongiu@unimore.it

Anno accademico 2018/19

Rappresentazione binaria

- Tutta l'informazione interna ad un computer è codificata con sequenze di due soli simboli: 0 e 1
  - è facile realizzare dispositivi elettronici che discriminano fra due stati, molto meno se gli stati sono tanti
- L'unità elementare di informazione si chiama bit da 'binary digit'
- Byte : sequenza di 8 bit

Sistema decimale posizionale (1)

- Un numero (es. 5) può essere rappresentato in molti modi:

  - cinque, five, 5, V, XXXXX …..

- Rappresentazioni diverse hanno proprietà diverse
  - moltiplicare due numeri in notazione romana è molto più difficile che moltiplicare due numeri in notazione decimale ….
- Noi siamo abituati a lavorare con numeri rappresentati in notazione posizionale in base 10

Sistema decimale posizionale (2)

- La rappresentazione di un numero intero in base 10 è una sequenza di cifre scelte fra l'insieme {0 1 2 3 4 5 6 7 8 9}
- Il valore di una rappresentazione

$$a_N a_{N-1} \ldots a_0 \ , \ a_{-1} a_{-2} \ a_{-3} a_{-4} \ldots$$

è dato da

$$a_N \cdot 10^N + a_{N-1} \cdot 10^{N-1} \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0$$
$$+$$
$$a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} + a_{-3} \cdot 10^{-3} + a_{-4} \cdot 10^{-4} + \ldots$$

- b= 10 è la base
- $\mathbf{10^i}$ è il peso della cifra $\mathbf{a_i}$ nel valore del numero

- Sistema decimale posizionale

Sistema decimale posizionale (3)

- $253 = 2 \times 100 + 5 \times 10 + 3 \times 1 =$
  $= 2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0$

- $23{,}47 = 2 \times 10 + 3 \times 1 + 4 \times 0.1 + 7 \times 0.01 =$
  $= 2 \times 10 + 3 \times 1 + 4 \times (1/10) + 7 \times (1/100) =$
  $= 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$

- Sistema *posizionale* (romani: sistema non posizionale)
- Sistema *decimale* (maya: sistema non decimale)

Sistema decimale posizionale (4)

Alcune proprietà di questa notazione :

- Il massimo numero rappresentabile con N cifre è 99….9 (N volte 9, la cifra che vale di più), pari a 10^N-1
  - es: su tre cifre il massimo numero rappresentabile è 999 pari a 10^3-1 =1000-1

- Quindi se voglio rappresentare K diversi numeri (cioè *0 1 2 …K-1*) mi servono almeno almeno *x* cifre dove 10^x è la più piccola potenza di 10 che supera K
  - es: se voglio 25 configurazioni diverse mi servono almeno 2 cifre perché 10^2=100 è la più piccola potenza di 10 maggiore di 25

Notazione posizionale in base 2 (1)

- La rappresentazione di un numero intero in base 2 è una sequenza di cifre scelte fra {0,1} :
  - es: 10, 110, 1
- Il valore di una rappresentazione

  $a_N \ldots a_0 , a_{-1} a_{-2}\ a_{-3} a_{-4}$
- è dato da

  $a_N \times 2^N + a_{N-1} \times 2^{N-1} \ldots . + a_1 \times 2^1 + a_0 \times 2^0 +$
  $a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} + a_{-4} \cdot 2^{-4} + \ldots$

- b= 2 è la base 2^i è il peso della cifra a_i nel valore del numero

Notazione posizionale in base 2 (2)

Esempi :

- 10 = $1*2^1 + 0*2^0 = 2$
- 110 = $1*2^2 + 1*2^1 + 0*2^0 = 4 + 2 + 0 = 6$
- 1 = $1*2^0 = 1$

10 si legge “uno-zero” e non “dieci” !!!

Notazione posizionale in base 2 (3)

Per la base due valgono proprietà analoghe a quelle viste per la base 10 :

- Il massimo numero rappresentabile con N cifre è 11….1 (N volte 1, la cifra che vale di più), pari a 2^N-1
  - es: su tre cifre il massimo numero rappresentabile è 111 pari a 2^3-1 = 8 - 1 = 7
- Quindi se voglio rappresentare K diversi numeri (cioè *0 1 2 …K-1*) mi servono almeno almeno *x* cifre dove 2^x è la più piccola potenza di 2 che supera K
  - es : se voglio 25 configurazioni diverse mi servono almeno 5 cifre perché 2^5=32 è la più piccola potenza di 2 maggiore di 25

Conversione di interi : Base 10 → Base 2

- Successione di divisioni per 2 :
  - termina quando il resto è 0
- La conversione in binario si ottiene leggendo i resti determinati in ordine inverso

es.: 13	6	3	1	0	Quozienti
	1	0	1	1	Resti

$13_{10} = \mathbf{1101_2}$

Conversione di interi Base 2 → Base 10

- Somma pesata delle cifre binarie:

es.: $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 8 + 4 + 0 + 1$

$= \mathbf{13}_{10}$

Numeri binari interi: esempi

0	**0**	8	**1000**	16	**10000**
1	**1**	9	**1001**	17	**10001**
2	**10**	10	**1010**	18	**10010**
3	**11**	11	**1011**		…..
4	**100**	12	**1100**		
5	**101**	13	**1101**		
6	**110**	14	**1110**		
7	**111**	15	**1111**		

Numeri binari interi: esempi(2)

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$

$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
$2^{11} = 2048$
$2^{12} = 4096$
...
$2^{16} = 65536$
...
$2^{24} \cong$ 16 milioni
...

Aritmetica binaria

- Necessità di codificare nel mondo dei numeri binari ogni operazione aritmetica.

- addizione: es.:

  0+0=0
  0+1=1
  1+0=1
  1+1=0 col riporto di 1

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 0101 + \\ 0011 = \\ \hline 1000 \end{array} \qquad \begin{array}{r} 5_{10} + \\ 3_{10} = \\ \hline 8_{10} \end{array}$$

- Sottrazione

  0-0=0
  0-1=1 col prestito di 1 dalla cifra precedente
  1-0=1
  1-1=0

Aritmetica binaria (2)

- Moltiplicazione:
- es.: per 2, 2^2, 2^3, ... $\leftrightarrow$ ‘shift’ (traslazione) verso sx di 1, 2, 3 bit

  1101 x 100 = 110100

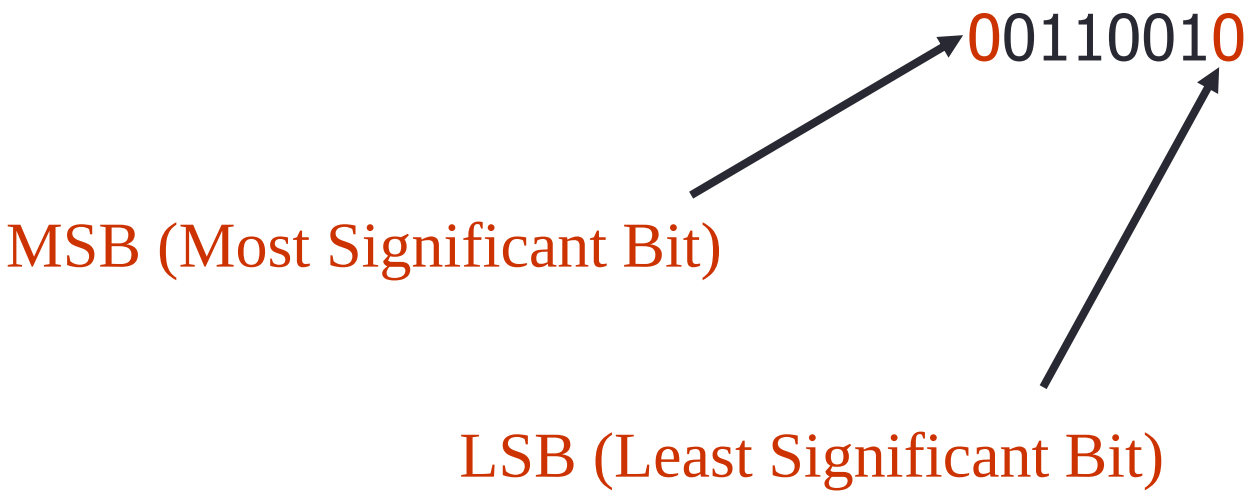
  (13 x 4 = 52)

La rappresentazione dei numeri all'interno di un computer

- Usa la notazione binaria
- Ogni numero viene rappresentato con un numero finito di cifre binarie (*bit*)
- Numeri di 'tipo' diverso hanno rappresentazioni diverse
  - es. interi positivi, interi (pos. e neg.), razionali, reali, complessi

La rappresentazione dei numeri all'interno di un computer (1)

- Alcuni termini utili:
  - *byte* : una sequenza di 8 bit
  - *word* (parola) : 2, 4, 8 byte (dipende dalla macchina) unità minima che può essere fisicamente letta o scritta nella memoria
- Tipicamente gli interi positivi si rappresentano usando 2 o 4 byte
- Notazione



La rappresentazione dei numeri all'interno di un computer (2)

- Alcuni punti importanti:
  - se uso 4 byte (32 bit) posso rappresentare solo i numeri positivi da 0 a 2^{32}-1, che sono molti ma non *tutti* !
  - se moltiplico o sommo due numeri molto elevati posso ottenere un numero che non è rappresentabile
    - es: vediamo cosa succede in base 10 con solo 3 cifre :

      500 + 636 = 1136 risultato 136

    se uso solo 3 cifre non ho lo spazio fisico per scrivere la prima cifra (1) che viene 'persa', è un fenomeno chiamato *overflow*

101 +

110 =

1011

il primo 1 non trova spazio

La rappresentazione dei numeri all'interno di un computer (3)

- Interi positivi e negativi :
  - ci sono diverse convenzioni di rappresentazione
    - *modulo e segno* in cui il primo bit viene riservato al segno (1 negativo, 0 positivo) e gli altri 31 al modulo
    - *Complemento a due*
    - *Complemento a uno* (la trascuriamo)
  - rimane comunque il problema dell'overflow

La rappresentazione dei numeri all'interno di un computer (4)

Numeri relativi

- Modulo e segno (es con 3 bit)

0 segno +
1 segno -

- codifica semplice
- operazioni aritmetiche più complesse

es.: +2 ↔ 010 e -2 ↔ 110

001 +	1 +
110 =	-2 =
111	-3

Errato!

Occorre differenziare tra i bit del numero e quelli di segno
Bisogna codificare in modo diverso le operazioni aritmetiche.

La rappresentazione dei numeri all'interno di un computer (5)

- Complemento a due (es con 4 bit)

  es: +5 = 0101 -5 ??
- Partendo da +5 = 0101 si invertono gli 1 con gli 0 : 1010
- Si aggiunge 1: 1010 + 1 = 1011 = -5

  - 1 x 2^3 +0 x 2^2 + 1 x 2^1 + 1 x 2^0

  = - 8 +0 + 2 + 1 = -5

Il primo bit non rappresenta solo il segno!

Non occorre più pertanto differenziare i bit.

Esempio

0001 +	1 +
1011 =	-5 =
1100	-4

OK!

Altre basi numeriche utilizzate

- Ottale (base8): { 0, 1, 2, 3, 4, 5, 6, 7} (10↔8)
- Esadecimale (base 16): { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  (10↔16)
- Usate perché semplici conversioni da base 2 a base 8 o 16:

Esempio: $11100011010 1_2$

$\underbrace{111}\ \underbrace{000}\ \underbrace{110}\ \underbrace{101} = 7065_8$

$\underbrace{1110}\ \underbrace{0011}\ \underbrace{0101} = E35_{16}$

Ulteriore sistema di codifica dei numeri

BCD (Binary-Coded Decimal)

- Si codificano in binario (4 bit) le singole cifre decimali.
- es.: 254

- nessun errore di conversione
- precisione dei calcoli decimali
- spreco di cifre
- usato nelle calcolatrici tascabili

La rappresentazione dei numeri all'interno di un computer (6)

- Razionali
  - numero finito di cifre periodiche dopo la virgola (ad esempio 3.12 oppure 3.453
  - rappresentazione solitamente su 4/8 byte
  - rappresentazione in *virgola fissa* : riservo X bit per la parte frazionaria

  - es : con 3 bit per la parte intera e 2 per quella frazionaria 011.11, 101.01

La rappresentazione dei numeri all'interno di un computer (7)

- Come si converte in base 10 una rappresentazione in virgola fissa
  - es :

$101.01 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} =$

$= 4 + 1 + 0.25 = 5.25$

dove $2^{-1} = 1/2 = 0.5$, $2^{-2} = 1/2^2 = 0.25$

e in generale $2^{-n} = 1/2^n$

La rappresentazione dei numeri all'interno di un computer (8)

- Problemi della rappresentazione in virgola fissa
  - overflow
  - underflow quando si scende al di sotto del minimo numero rappresentabile
    - es. vediamo in base 10, con 2 cifre riservate alla parte frazionaria 0.01 / 2 = 0.005 non rappresentabile usando solo due cifre

La rappresentazione dei numeri all'interno di un computer (9)

- Problemi della rappresentazione in virgola fissa (cont.)
  - spreco di bit per memorizzare molti '0' quando lavoro con numeri molto piccoli o molto grandi
    - es. vediamo in base 10, con 5 cifre per la parte intera e 2 cifre riservate alla parte frazionaria

    10000.00 oppure 00000.02
  - i bit vengono usati più efficientemente con la notazione *esponenziale* o *floating point* (*virgola mobile*)

La rappresentazione dei numeri all'interno di un computer (10)

- Rappresentazione in virgola mobile
  - idea : quando lavoro con numeri molto piccoli uso tutti i bit disponibili per rappresentare le cifre dopo la virgola e quando lavoro con numeri molto grandi le uso tutte per rappresentare le cifre in posizioni elevate
  - questo permette di rappresentare numeri piccoli con intervalli minori fra loro rispetto ai numeri grandi
  - questo riduce gli errori nel calcolo a parità di bit utilizzati

La rappresentazione dei numeri all'interno di un computer (11)

- Rappresentazione in virgola mobile (cont.)
  - ogni numero N è rappresentato da una coppia
    (*mantissa* M, *esponente* E) con il seguente significato

    $$N = M * 2^E$$
  - esempi:
    1. in base 10, con 3 cifre per la mantissa e 2 cifre per l'esponente riesco a rappresentare

       $$349\,000\,000\,000 = 3{,}49 * 10^{11}$$

       con la coppia (3.49,11) perché M = 3.49 ed E = 11

La rappresentazione dei numeri all'interno di un computer (12)

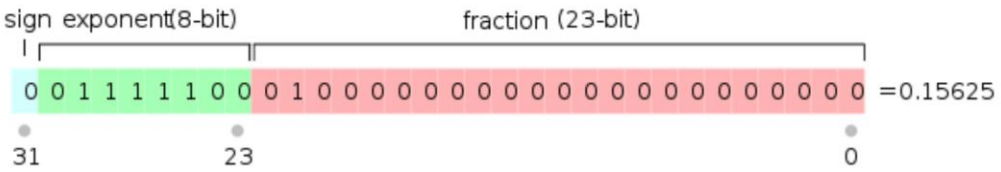
- Rappresentazione in virgola mobile (cont.)
  - in base 10, con 3 cifre per la mantissa e 2 per l'esponente riesco a rappresentare 0.000 000 002 = $2.0 * 10^{-9}$ con la coppia (2.0,-9) perché M = 2.0 ed E = -9
  - sia 0.000 000 002 che 349 000 000 000 non sono rappresentabili in virgola fissa usando solo 5 cifre decimali

Lo standard IEEE 754

- Insieme di rappresentazioni di valori numerici e simbolici usato per floating point computation attraverso software (librerie) o hardware
- Vari tipi di precisione
- Float e double in C seguono questo standard
- Il numero viene formato con un bit di segno, con un esponente e con la mantissa
- Si specificano 3 parametri:
  - P: precisione o numero di bit che compongono la mantissa
  - E_{max}: esponente massimo
  - E_{min}: esponente minimo
- Ad esempio per la precisione singola P=23, E_{max}=127 e E_{min}=-126
- La mantissa viene normalizzata scegliendo l'esponente in modo che sia sempre nella forma 1,xxxx...
- L'esponente è polarizzato, ovvero ci si somma E_{max} (costante di polarizzazione)

Esempio

- 0.15625_{10} in binario diventa 0.00101_2 (i.e., 1/8 + 1/32)
- $0.00101_2 = 1.01_2 \times 2^{-3}$
- Mantissa frazionaria: $.01_2$
- Esponente: -3
- Esponente polarizzato (precisione singola): -3 + 127 = 124



- Esponente polarizzato (precisione doppia): -3 + 1023 = 1020

Lo standard IEEE 754

- Nel caso della precisione singola usare le seguenti relazioni:

	esp	M	numero
Numero normalizzato	0<esp<255	qualsiasi	$(-1)^s\ (1,M)\ 2^{esp-127}$
Numero denormalizzato	esp=0	M ≠ 0	$(-1)^s\ (\underline{\mathbf{0}},M)\ 2^{-126}$ Riduce la perdita di precisione se underflow
Zero	esp=0	M = 0	$(-1)^s\ 0$
Infinito	esp=255	M = 0	$(-1)^s\ \infty$
NaN (Not a Number)	esp=255	M ≠ 0	*NaN*

Range e precisione

- I numeri piu' piccoli (vicini allo zero) rappresentabili
- Esp=1, M=0 → $\pm 2^{-126} \approx \pm 1.17549 \times 10^{-38}$ (norm. singola)
  $\pm 2^{-1022} \approx \pm 2.22507 \times 10^{-308}$ (norm. doppia)
- Esp=0, M=1 → $\pm 2^{-149} \approx \pm 1.40130 \times 10^{-45}$ (denorm. singola)
  $\pm 2^{-1074} \approx \pm 4.94066 \times 10^{-324}$ (denorm. doppia)

- I numeri finiti piu' grandi (lontani dallo zero) rappresentabili a singola precisione sono
- Esp=254, M=11…1 → $\pm(1-2^{-24}) \times 2^{128} \approx \pm 3.40282 \times 10^{38}$

- A doppia: $\pm 1.79769 \times 10^{308}$

Esempio

- Voglio rappresentare il numero -36,47 usando la convenzione IEEE754, ovvero vedere come viene realmente memorizzata la variabile

```
float f=-36.47
```

- Prima di tutto calcolo la rappresentazione binaria. A tal fine calcolo parte intera e frazionaria mediante iterazione di divisioni/moltiplicazioni per 2.

esempio (2)

36 div 2	(resto)
18	0
9	0
4	1
2	0
1	0
0	1

0,47 x 2	0	,94
0,94 x 2	1	,88
0,88 x 2	1	,76
0,76 x 2	1	,52
0,52 x 2	1	,04
0,04 x 2	0	,08
…	…	…

$-36.47_{10} = -100100{,}011110\ldots_{2} = -1{,}0010001111 \times 2^{5}$

Ora ho tutti gli elementi da collocare nella rappresentazione.

Rappresentazione di un insieme finito di oggetti

- Vogliamo rappresentare i giorni della settimana :
  - {Lu, Ma, Me, Gio, Ve, Sa, Do}
  - usando sequenze 0 e 1
- Questo significa costruire un ‘codice’, cioè una tabella di corrispondenza che ad ogni giorno associa una opportuna sequenza
- In principio possiamo scegliere in modo del tutto arbitrario….

Rappresentazione di un insieme finito di oggetti

- Una possibile codifica binaria per i giorni della settimana

Lunedì	0100010001
Martedì	001
Mercoledì	1100000
Giovedì	1
Venerdì	101010
Sabato	111111
Domenica	000001

Rappresentazione di un insieme finito di oggetti

- Problema : la tabella di corrispondenza fra codifiche tutte di lunghezza diversa
  - spreco di memoria
  - devo capire come interpretare una sequenza di codifiche
  - 110000011 = Me Gio Gio
  - 110000011 = Gio Gio Do Gio
- Soluzione

  si usa un numero di bit uguale per tutti : il minimo indispensabile

Rappresentazione di un insieme finito di oggetti

- Per rappresentare 7 oggetti diversi servono almeno 3 bit (minima potenza di due che supera 7 è 8= 2^3) quindi :
  - 000 Lunedì
  - 001 Martedì
  - 010 Mercoledì
  - 011 Giovedì
  - 100 Venerdì
  - 101 Sabato
  - 110 Domenica
  - 111 non ammesso

Rappresentazione di caratteri e stringhe

- Tipologia di caratteri:
  - alfabeto e interpunzioni: A, B, .., Z, a, b, .., z, ;, :, “, ..
  - cifre e simboli matematici: 0, 1, ..., 9, +, -, >, ..
  - caratteri speciali: £, $, %, ...
  - caratteri di controllo: CR, DEL, ...
- Le stringhe sono sequenze di caratteri terminate in modo particolare.
- I caratteri sono un insieme finito di oggetti e seguono la strategia vista per i giorni della settimana

Rappresentazione di caratteri e stringhe (2)

- **ASCII** (*American Standard Code for Information Interchange*): Codice a 7 bit (standard)
- **ASCII** esteso a 8 bit (non standard)
- es.: A 01000001
  (00101000
- **UNICODE**: su 16 bit (65536 diverse configurazioni): più recente, permette di rappresentare anche alfabeti diversi e simboli per la scrittura di lingua orientali.

Rappresentazione di caratteri e stringhe (3)

- **ASCII a 7 bit**

I 7 bit sono suddivisi logicamente in due campi rispettivamente di 3 e 4 bit.

I primi tre bit rappresentano categorie di caratteri, mentre gli ultimi quattro servono a rispettare l' ordinamento dei caratteri all' interno di ogni categoria.

Rappresentazione di caratteri e stringhe (4)

- Categorie

1°bit	2°bit	3°bit	Caratteri rappresentati
0	1	0	simboli di punteggiatura, simboli speciali e di operazione
0	1	1	numerali
1	0	0	maiuscole (A - O)
1	0	1	maiuscole (P - Z)
1	1	0	minuscole (a - o)
1	1	1	minuscole (p - z)

Rappresentazione di caratteri e stringhe (5)

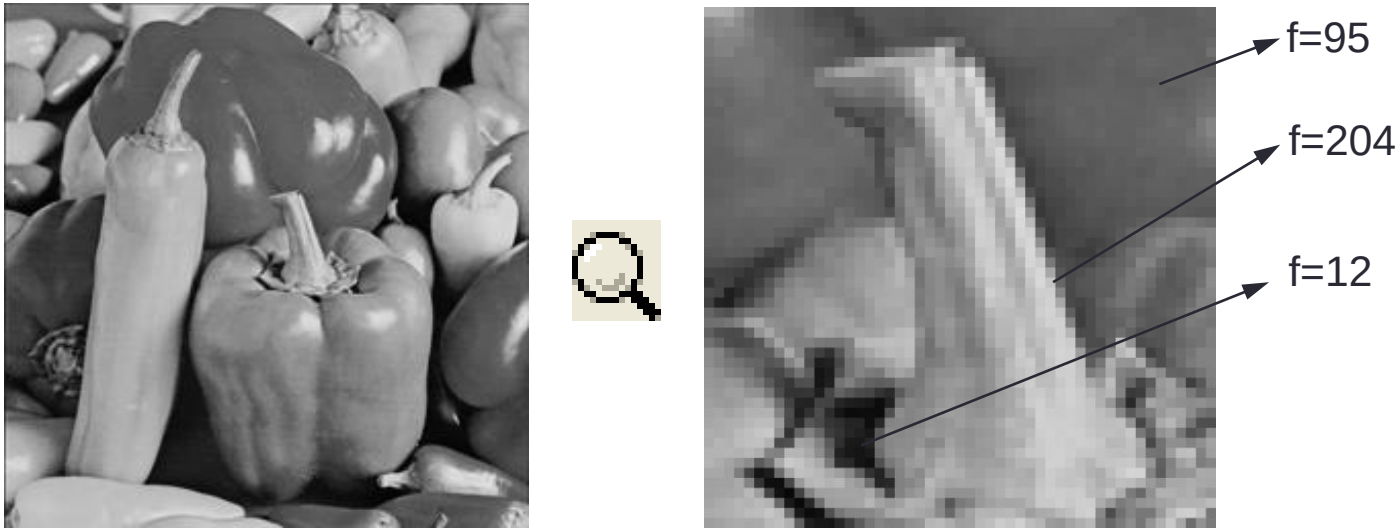
- 4°, 5°, 6°, 7° bit
- Nei numerali sono costituiti dalla codifica in binario su 4 bit delle cifre decimali (codice BCD).
- Per i caratteri dell'alfabeto la codifica è tale da rispettare l'ordinamento alfabetico
- Esempi

b (2ª lettera) : 1100010 B: 1000010

2

Cos'è un'immagine digitale

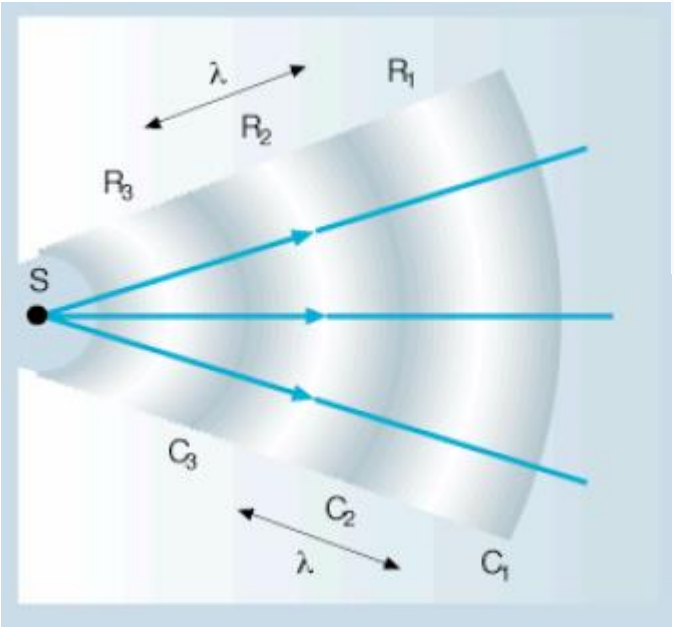
Un'**immagine digitale** può essere vista come una funzione bi-dimensionale *f(x,y)*, dove *f* rappresenta l'**intensità** o **livello di grigio** dell'immagine in quel punto: 0 rappresenta il **nero**, 255 il **bianco**

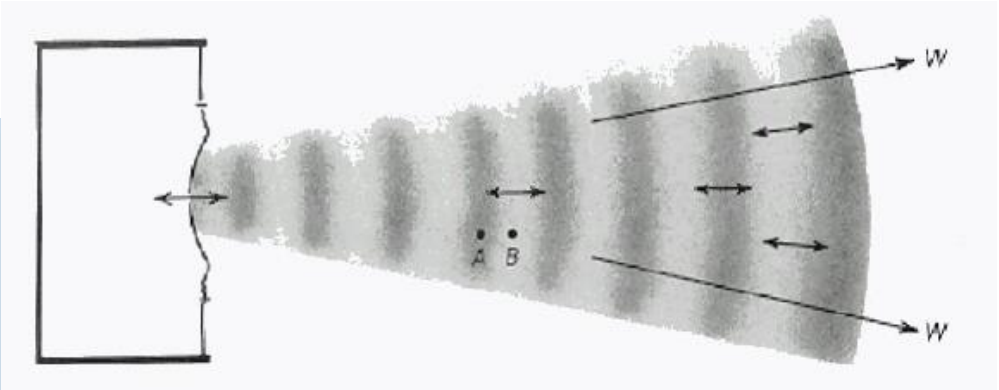


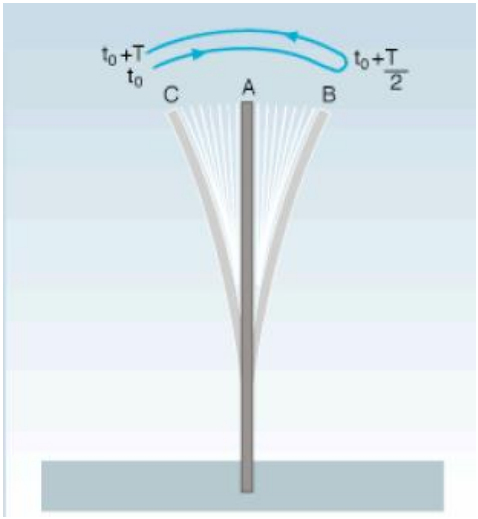
Un'*immagine* è quindi una **matrice** di elementi chiamati **pixels** (picture elements).

La natura fisica del suono

- onde che trasportano energia lontano dalla sorgente (oggetto che vibra)







Rappresentazione di audio

- E’ possibile rappresentare il suono con la sua forma d’onda
- Al calcolatore, basta rappresentare una opportuna sequenza di campioni della forma d’onda