Rappresentazione binaria delle informazioni
oppure
Rappresentazione digitale delle informazioni

Argomenti in dettaglio

- Struttura modulare a livelli
- Rappresentazione binaria informazioni
- Linguaggio macchina
  - Cosa è
  - Come si usa

4. Reti logiche combinatorie e sequenziali (Hardware)
5. Unità di elaborazione (Hardware e Firmware)
6. Strutture di interconnessione ed arbitri
7. Processore Sequenziale
   - Come funziona
   - Relazione con linguaggio macchina

- Gestione Interruzioni
- Gestione cache
- Gestione multiprogrammazione

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
- Si utilizza una rappresentazione digitale
- Questa forma è l'unica utilizzabile per quasiasi informazione memorizzata = digitalizzazione delle informazione
- La digitalizzazione è la ragione della potenza delle reti informatiche che possono memorizzare e trasmettere tutte le informazioni precedenti

Codifica digitale

- L'unità minimale di rappresentazione è il ***bit*** (binary digit – cifra digitale): **0** o **1**
- Informazioni complesse si memorizzano come sequenze di bit
- Una sequenza di **8 bit** viene chiamata ***Byte***
  - 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 1
  - .....

Codifica non ridondante dell'informazione

- Utilizzando una stringa **N** bit si possono codificare $\mathbf{2^N}$ informazioni (tutte le possibili combinazioni di 0 e 1 su N posizioni)
- Per codificare N informazioni diverse sono quindi necessari almeno $\lceil \log_2(N) \rceil$ = il più piccolo intero maggiore di $\log_2(N)$
- Con un byte si possono codificare quindi $\mathbf{2^8 = 256}$ possibili informazioni
- Per codificare 1000 informazioni servono $\lceil \log_2(1000) \rceil = \lceil 9.9\ldots \rceil = 10$ bit

Altre unità di misura

- **KiloByte (KB), MegaByte (MB), GigaByte (GB)**
- Per ragioni storiche in informatica Kilo, Mega, e Giga indicano le più piccole ***potenze di 2*** maggiori delle corrispondenti potenze di 10
- Più precisamente
  - 1 KB = 1024 x 1 byte = 2^{10} ~ 10^3 byte
  - 1 MB = 1024 x 1 KB = 2^{20} ~ 10^9 byte
  - 1 GB = 1024 x 1 MB =2^{30} ~ 10^{12} byte
  - ...
- I multipli del byte vengono utilizzati come unità di misura per la capacità delle memorie
- In realtà la singola posizione di memoria contiene 4 o 8 byte

La Codifica dei Caratteri

A B ... a b .... & % $ ...

Codici per i simboli dell'alfabeto

- Per rappresentare i simboli dell'alfabeto anglosassone (0 1 2 ... A B ... A b ...) bastano 7 bit
  - Nota: *B* e *b* sono simboli diversi

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

- Per manipolare un numero maggiore di simboli la Microsoft ha introdotto la codifica UNICODE a 32 bit (2^{32} caratteri)

Codifica ASCII

- La codifica ASCII (American Standard Code for Information Interchange) utilizza codici su 8 bit
- Ad esempio
  - 0 1 0 0 0 0 0 1 rappresenta A
  - 0 1 0 0 0 0 1 0 rappresenta B
  - 0 1 0 0 0 0 1 1 rappresenta C
- L'ordine alfabetico è rispettato se si considerao le codifiche come numeri e vedendo se un numero è maggiore di un altro
- Le parole si codificano utilizzando sequenze di byte
  - 01000010 01000001 01000010 01000001
    B A B A

La codifica dei numeri (interi relativi e reali)

Rappresentazione decimale

- La numerazione decimale utilizza una *codifica posizionale basata sul numero 10* e sull'alfabeto di simboli 0 1 2 ... 9 introdotta dagli arabi
- I numeri si leggono da sinistra a destra e le diverse cifre hanno un peso diverso che corrisponde a diverse potenze di 10 (mille, diecimila, ecc)
- Es. la sequenza `312' rappresenta il numero
  - $3 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$
- La notazione posizionale può essere utilizzata in qualsiasi altra *base*

Notazione posizionale in base B

Fissata una qualsiasi base B > 1 la sequenza

$c_n\ c_{n-1}\ \ldots\ c_1\ c_0$ dove ciascun $c_k < B$

rappresenta il numero

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

$$r = \sum_{i=0,.n} c_i \times B^i$$

B^i è indicata come la significatività di c_i . La stessa cifra in posizioni diverse ha una significatività diversa

Basi comunemente usate

- Base decimale **B = 10**: alfabeto 0,1,2,3,4,5,6,7,8,9
- Base binaria **B=2**: alfabeto 0,1
- Base ottale **B=8**: alfabeto 0,1,2,3,4,5,6,7
- Base esadecimale **B=16**: alfabeto 0,1,…,9,A,B,C,D,E,F

  dove A vale 10, B vale 11,…, F vale 15

Rappresentazione binaria

- Se B=2 la sequenza

$$c_{n-1}\, c_{n-2} \cdots c_1\, c_0$$

- Dove ciascun $c_k < 2$ rappresenta il numero

$$c_0 \times 2^0 + c_1 \times 2^1 + \ldots c_{n-1} \times 2^{n-1}$$

- la sequenza **1011** in base 2 denota il numero

$$1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 11$$

- la sequenza **1011** in base 4 denota il numero

$$1 \times 4^0 + 1 \times 4^1 + 0 \times 4^2 + 1 \times 4^3 = 69$$

Espressività

- Fissata una base B>1, con una sequenza di lunghezza K posso rappresentare B^K oggetti diversi, numeri nel nostro caso. Se rappresento naturali, ad esempio da 0 ... B^K – 1
- Se B=10 allora con 4 cifre posso rappresentare i numeri da 0 a 9999 (i primi10000 numeri)
- Se B=2 allora con 3 cifre posso rappresentare i numeri da 0 a 7 (i primi 8 numeri)
- Notare che posso rappresentare un qualunque numero in un intervallo di ampiezza B^K
  - 0.. B^K-1
  - 100..100+ B^K –1=99+ B^K
  - A..A-1+ B^K nel caso più generale, A anche negativo

Lunghezza delle rappresentazioni nel caso generale

- Fissata una base B>1 e dato un numero M, qual'è la lunghezza minima L per poter rappresentare tutto l'intervallo 0..M in base B?
- L deve soddisfare $B^L - 1 >= M$ dove $B^L - 1$ è il numero più grande rappresentabile in L cifre
- La soluzione è quindi $\lceil \log_B(M+1) \rceil$
- Es. B=2 e M=9, allora L=4;
  - infatti su 3 bit rappresento solo i numeri da 0...7;
  - su 4 bit rappresento i numeri da 0...15

Funzione di codifica

- Dato M la sua codifica posizionale in base B codice(M,B) si estrae utilizzando la divisione con resto come segue
  - $M = A_1 * N + B_0$ *(M diviso B con resto B_1)*
  - $A_1 = A_2 * N + B_1$
  - ....
  - $A_{k-1} = A_k * N + B_{k-1}$
- Fino a che il quoziente A_k diventa 0
- Infine si definisce

  codice(M,B) = $B_{k-1} \ldots B_0$

Esempio in base 2

- Rappresentare 13 in base 2
  - Servono almeno 4 cifre (2^3=8)
  - 13 : 2 = 6 con resto 1
  - 6 : 2 = 3 con resto 0
  - 3 : 2 = 1 con resto 1
  - 1: 2= 0 con resto 1
  - codice(13,2) = 1 1 0 1 = 8+4+1
- Rappresenta 51 in base 2
  - Servono almeno 6 cifre (2^6=64)
  - 51 : 2 = 25 con resto 1
  - 25 : 2 = 12 con resto 1
  - 12 : 2 = 6 con resto 0
  - 6: 2= 3 con resto 0
  - 3:2 = 1 con resto 1
  - 1:2= 0 con resto 1
  - codice(51,2) = 1 1 0 0 0 1 = 32+16+2+1

Esempio in base 8

- Come si rappresenta 140 in base 8?
- Ci servono almeno 3 cifre (8^2=64)
- 140 : 8 = 17 con resto 4
- 17 : 8 = 2 con resto 1
- 2:8 = 0 con resto 2
- Codice(140,8) = 214 = $2*8^2 + 8 + 4$

Esempio in base 16

- Come si rappresenta 140 in base 16?
- Ci servono almeno 2 cifre
- 140 : 16 = 8 con resto 12 rappresentato con C
- 8:16= 0 con resto 8
- Codice(140,16) = 8C = 8*16 + 12

Cambio di base tra potenze del due

- Le basi del tipo 2^k hanno una proprietà interessante per quanto riguarda il passaggio da una rappresentazione all'altra
- Ad es. per passare dalla base 2 alla base 8 basta raggruppare gruppi di 3 bit e trasformarle in cifre in base 8
- Per passare dalla base 8 alla base 2 basta espandere le cifre nelle corrispondenti codifiche binarie su 3 bit

001 101 (binario)
1 5 (ottale)

Metodo diverso per conversione

A in base B

$= c_0 \times B^0 + c_1 \times B^1 + \ldots + c_{n-1} \times B^{n-1} = \Sigma_{i=0,.n-1}\ c_i \times B^i$

dove

- n è il più piccolo intero tale per cui $B^n > A$
- c_{n-1} è il quoziente della divisione di A per B^{n-1} sia Rn il resto
  $= \Sigma_{i=0,.n-2}\ c_i \times B^i$
- c_{n-2} è il quoziente della divisione di Rn per B^{n-2} oppure
  c_{n-2} è il quoziente della divisione di 2*Rn per B^{n-1}
- …

In questo modo vengono generati i valori in ordine inverso a quello del metodo precedente

= le cifre sono generate a partire da quella più significativa

Esempio

Convertire 85 in base 2

- Servono 7 cifre $2^6<85<2^7$
- 85:64= 1 resto 21 prima cifra 1
- 21*2=42:64 0 resto 42 seconda cifra 0
- 42*2=84:64 1 resto 20 terza cifra 1
- 20*2=40:64 0 resto 40 quarta cifra 0
- 40*2=80:64 1 resto 16 quinta cifra 0
- 16*2=32:64 0 resto 32 sesta cifra 1
- 32*2=64:64 1 resto 0 settima cifra 1

1010101=64+16+4+1

Nel caso di base due non è necessaria la divisione, basta la sottrazione

Operazioni su numeri binari

- La codifica in binario dei numeri naturali permette di utilizzare operazioni `bit per bit' per costruire operazioni su sequenze quali la somma
- Operazione di somma su un bit
  - 0 + 0 = 0 0 + 1 = 1 1 + 0 = 1
- 1 + 1 non è rappresentabile su un bit
- Su più bit allora 1+1 = 0 e genera un riporto di 1

Quindi servono due funzioni, una genera la somma e l'altra il riporto

	S	R
0+0	0	0
1+0	1	0
0+1	1	0
1+1	0	1

La presenza del riporto costringe ad usare funzioni o tabelle con tre input, due per gli addendi ed una per il riporto

Addizione

- Si utilizza la somma bit per bit propagando il riporto (come nei decimali)

Binario	Decimale
– 0 1 1 0 1 +	13 +
– 0 1 0 0 1 =	9 =
– 1 0 1 1 0	22

- Un'altra operazione semplice è la moltiplicazione per 2 si aggiunge uno zero in fondo a destra (come in decimale la moltiplicazione per 10)
  - 0 1 1 0 1 (13) diventa 1 1 0 1 0 (26)
- Moltiplicazione si può svolgere come nel caso decimale

```
100110 * 101                100110
                                101
                            100110
                           000000 -
                          100110 -
                    ------------------
                          10111110
```

Numeri relativi

- Esistono vari metodi per rappresentare numeri sia positivi che negativi
- L'obiettivo è comunque quello di ottenere algoritmi semplici per costruire le operazioni aritmetiche direttamente con operazioni sui bit
- Esempi di possibili rappresentazioni:
  - Bit di segno (Modulo e segno)
  - Complemento a 1
  - Complemento a 2

Rappresentazione con bit di segno

- Fissato il numero di bit, il primo bit a sinistra identifica il segno
  - Su un byte (8 bit):

    00000001 = 1 10000001 = -1
- Con N bit a disposizione si codifica da $-(2^{N-1}-1)$ a $2^{N-1}-1$
- L'operazione di somma è basata tuttavia sull'analisi dei casi possibili a seconda del segno degli operandi
- Purtroppo non si riduce in modo semplice ad operazioni bit a bit quindi non si possono costruire componenti elettronici perchè in base a segno operazione di somma può diventare di sottrazione o viceversa
- Test di interesse
  - Uguale a zero = primo bit qualsiasi, tutti altri uguali a zero
  - Maggiore di zero = primo bit zero, almeno uno degli altri diverso da zero
  - Maggiore o uguale a zero = primo bit a zero oppure tutti gli altri a zero
  - Minore di zero = primo bit a uno, almeno uno degli altri diverso da zero

- Fissato il numero di bit utilizziamo invece la seguente rappresentazione

$$r = (-1)^{*}c_{n-1} \times (B^{n-1}-1) + \Sigma_{i=0,.n-2}\, c_i \times B^i$$

- Ad esempio
  - 0 0000010 = 2
  - 1 1111101 = -2
- Con $-(2^{N-1}-1)$ a $2^{N-1}-1$

Per rappresentare A

- Data la rappresentazione binaria su n cifre del valore assoluto di A
- Si aggiunge uno zero prefisso se positivo
- Si aggiunge uno zero prefisso e poi si invertono tutti i bit nel caso di numero negativo

Complemento a 1: somma

- Somma bit per bit? Funziona quasi sempre

00110+ (+6)	**11001+ (-6)**
11010= (-5)	**11010= (-5)**
00000 (0)	**10011 = (-12)**

- Se gli operandi hanno segno negativo si ottiene il risultato decrementato di 1
  - Es. –12 invece di –11
  - Occorre quindi aggiungere 1 quando si sommano numeri che danno un risultato negativo= si genera riporto sulle ultime cifre
- Anche in questo caso lo zero ha due rappresentazioni:

  0=00000000 0=11111111
- Test di interesse
  - Uguale a zero = tutti bit a uno oppure tutti a zero
  - Maggiore di zero = primo bit zero, almeno uno degli altri diverso da zero
  - Minore di zero = primo bit a uno, almeno uno degli altri diverso da uno

Rappresentazione in complemento a 2

- Fissato il numero di bit consideriamo la seguente nuova codifica
  $r = (-1)^{*}c_{n-1} \times B^{n-1} + \Sigma_{i=0,..n-2}\ c_i \times B^i$
- Ad esempio su 8 bit
  0 0000010 = 2 1 1111110 = 1 1111101 + 1=comp(0 0000010) -2
- -2^{N-1} a $2^{N-1}-1$ (non ridondante)
- Data la rappresentazione binaria su n cifre del valore assoluto di A
  - Si aggiunge uno zero prefisso se A positivo
  - Si aggiunge uno zero prefisso e poi si invertono tutti i bit nel caso di A negativo a partire da 1 meno significativo (escluso)
- Test di interesse
  - Uguale a zero = tutti bit uguali a zero
  - Maggiore di zero = primo bit zero, almeno uno diverso da zero
  - Maggiore o uguale di zero= primo bit a zero
  - Minore di zero = primo bit a uno

- Somma bit per bit? Funziona sempre

  11010+ (-6) dove 6=0110 → 1001 + 1 = 1010

  11011= (-5) dove 5= 0101 → 1010 + 1 = 1011

  10101 = (-11) dove 11= 1011 → 0100+1 = 0101

- Inoltre lo zero ha una sola rappresentazione:
  - 00000000
- E' una buona rappresentazione
  - Rappresentazione unica
  - Test semplici per confronto con zero

Overflow o supero

- Qualunque rappresentazione finita di numeri, in una qualsiasi base ha il problema del supero = un numero il cui modulo è troppo grande per essere rappresentato con le cifre a disposizione
- Il problema è particolarmente importante nel caso di risultato di operazioni poiché anche se i numeri di partenza possono essere rappresentati, il modulo del risultato può impedire una corretta rappresentazione
- Per questo occorre inserire dei controlli per essere sicuri che non vi siano errori dovuti ad overflow
- Tutti i componenti elettronici ed i programmi devono comprendere questi controlli
- Ad esempio nei casi precedenti, il supero viene spesso rilevato scoprendo un segno del risultato incongruente con i valori di partenza = la somma di due positivi che da un numero negativo o viceversa

Numeri razionali

- Utilizzando opportune convenzioni possiamo pensare di rappresentare non solo interi ma anche razionali
  - Virgola fissa: si fissa il numero di cifre della parte decimale
  - Virgola mobile: si rappresentano esponente e mantissa
  - (Virgola= notazione all'inglese!)
  - Poichè per rappresentare un numero razionale può essere necessaria una sequenza infinita di simboli (che dipende dalla base scelta) in generale un numero razionale può essere rappresentato solo in modo approssimato

Virgola fissa

- Fissiamo quante cifre intere e quante decimali vogliamo rappresentare ed utilizziamo
  - potenze di 2 sia *positive* che *negative*!
- Ad esempio: se la cifra piu' a destra rappresenta ½ ($=2^{-1}$):
  - 10001 rappresenta 8.5 = 8 + ½
  - cioe' va letto come: 1000.1
- Si rappresentano in modo esatto solo numeri esprimibili come somme negative di potenze di 2 (occorre approssimare gli altri valori)
- Ci muoviamo all'interno di un intervallo fissato e con precisione nota

Virgola mobile

- Per rappresentare sia numeri molto piccoli che molto grandi si utilizza una rappresentazione in cui la virgola decimale varia a seconda del numero
- Si usa una rappresentazione del tipo:
  - Valore= Segno* $2^{+/-\text{ Esponente}}$*Mantissa
  - La mantissa viene normalizzata per ottenere una rappresentazione unica (varia tra ½ e 1).
- Cioe' fissata la base dobbiamo memorizzare su K bit le informazioni su: Segno Esponente Mantissa
- Poichè il primo bit della mantissa è sempre 1 non viene memorizzato
- Quando si opera sui numeri essi devono essere allineati

Standard IEEE

- *Precisione singola* su 32 bit
  - 1 bit di segno
  - 8 di esponente (da -126 a +127) (-126+valore memorizzato)
  - 23 di mantissa
  - Si possono rappresentare valori fino a 2 elevato a (-150)
- *Precisione doppia* su 64 bit
  - 1 bit di segno
  - 11 di esponente (da -1022 a +1023)
  - 52 di mantissa
  - Si possono rappresentare valori fino a 2 elevato a (-1075)

Overflow e underflow

- Anche la rappresentazione in virgola fissa e mobile, poiché è finita ha il problema dell'overflow
- Nella rappresentazione di numeri razionali sorge inoltre il problema dell'underflow, cioè di un numero troppo piccolo per essere rappresentato
- Questo problema sorge in modo particolare quando si lavora contemporaneamente su numeri i cui moduli differiscono per diversi ordini di grandezza perché i numeri di modulo più piccolo o le differenze tra numeri più piccole non influenzano il risultato
- Ad esempio
  - se sommo numeri molto grandi e numeri molto piccoli la somma considera solo i numeri molto grandi
  - una sottrazione tra numeri molto simili darà come risultato zero perché la differenza è troppo piccola per essere rappresentata (mantissa≠0, esponente negativo e modulo troppo grande)
- Il secondo caso è un underflow perché il numero è troppo piccolo per essere rappresentato (o perché esponente troppo grande) in modulo

Codifiche ridondanti

- Si possono usare codifiche ridondanti, con più bit di quelli necessari per scoprire errori dovuti a fenomeni fisici (influenze elettriche, magnetiche)
- La codifica ridondante più nota è quella del bit di parità
  - Si codifica un valore con il minimo numero di bit
  - Si genera la codifica effettiva aggiungendo un bit in modo che il numero di bit complessivo a uno sia sempre
    - Pari = parità pari
    - Dispari = parità dispari
  - Se durante l'elaborazione si incontra una stringa con una parità sbagliata c'è stato un errore di elaborazione o trasmissione
- Esempio
  - n=48 6 bit necessari = 110000 con parità pari la stringa memorizzata è 1100000 in cui l'ultimo bit è quello di parità
  - n=49 110001 con parità pari la stringa memorizzata è 1100011 con parità uguale a 1

Codici ridondanti

- Codici rilevatori = permettono di scoprire un errore
- Codici correttori = permettono di scoprire e correggere errore
- La parità è un codice rilevatore di un errore singolo, infatti se vi sono due bit che cambiano di valore la parità non cambia
- Per scoprire e correggere un errore su m bit servono almeno r bit, dove

$$2^r \geq m+r+1$$

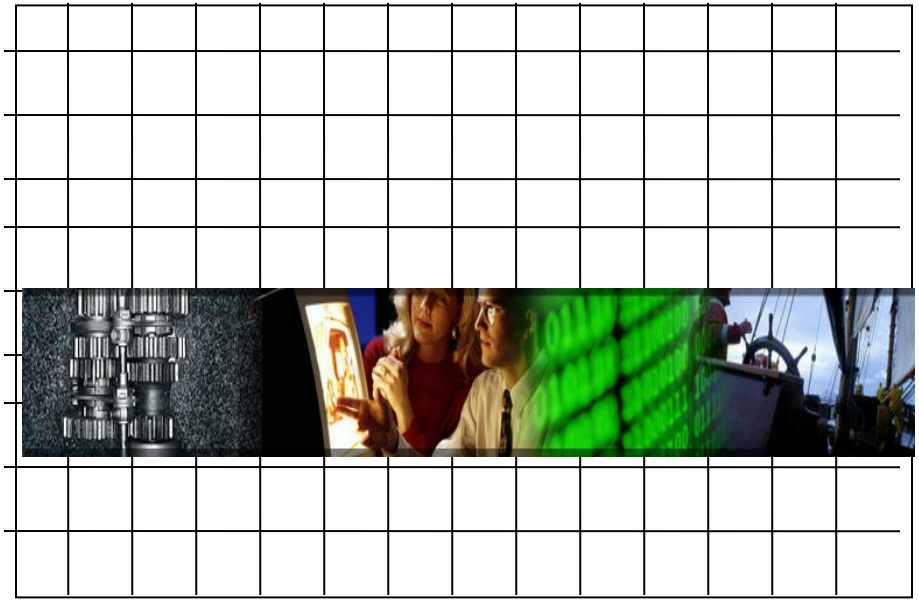
- Codice di Hamming
  - distanza di Hamming = numero di bit diverso in due stringhe, d(100110, 110010)=2
  - Devo codificare dei numeri, per correggere errori di d bit
    - scelgo delle codifiche che abbiano almeno distanza 2d+1 tra di loro
    - un errore di un bit genera una stringa che ha distanza al più d da quella originaria e d+1 dalle altre
    - posso rilevare e correggere errore
    - Per definire il codice reale, occorre definire bit di parità su sottostringhe con intersezione in modo da definire un sistema con condizioni

Codifica di immagini

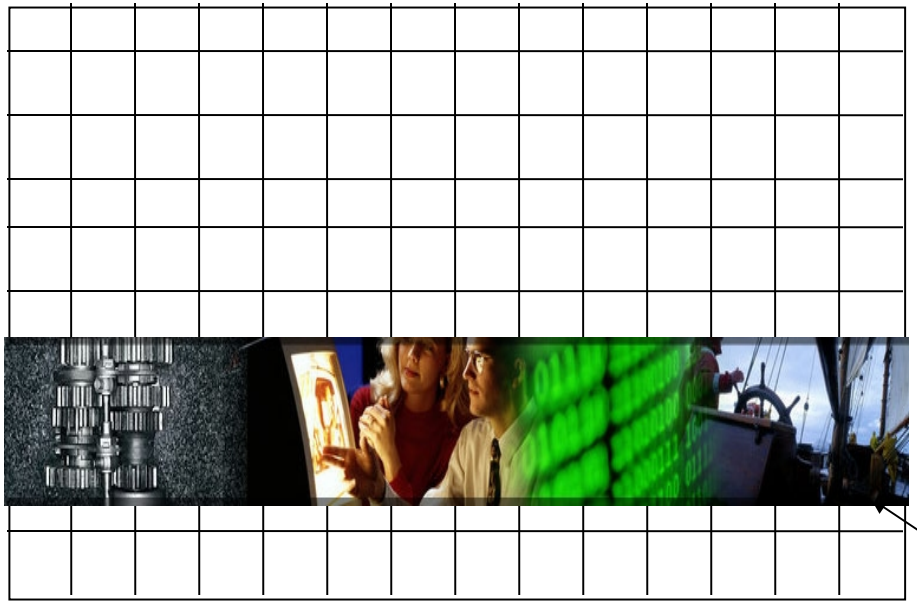
Pixel – Picture element

- Le immagini vengono scomposte in griglie
- Le caselle di una griglia vengono chiamate *pixel*
- La risoluzione indica il numero di pixel in cui è suddivisa un'immagine
  - Risoluzione tipica di uno schermo video 800 x 600, 1024 x 768

Codifica di pixel



Bitmap



Immagini in toni di grigio

- Se si assegna un solo bit a ogni pixel si rappresentano immagini in bianco e nero
  - 0 = bianco 1 = nero
- Per poter rappresentare immagini più complesse
  - si codificano i toni di grigio
  - Si associa una codifica di un tono di grigio ad ogni pixel

Immagini a colori

- Nella codifica RGB si utilizzano tre colori
  - rosso (Red), verde (Green) e blu (Blue):
- Ad ogni colore si associa un certo numero di sfumature codificate su N bit (2^N possibili sfumature)
- Ad esempio
  - se si utilizzano 2 bit per colore
    - si ottengono 4 sfumature per colore
    - ogni pixel ha un codice di 6 bit
  - Se si utilizzano 8 bit per colore
    - si ottengono 256 sfumature
    - 256^3 (16 milioni) possibili colori

Bitmap

- La rappresentazione di un'immagine mediante la codifica a pixel viene chiamata *bitmap*
- Il numero di byte richiesti per memorizzare una bitmap dipende dalla risoluzione e dal numero di colori
- Es. se la risoluzione è 640x480 con 256 colori occorrono 2.457.600 bit = 307 KB
- I formati bitmap più consciuti sono BITMAP (.bmp), GIF (.gif), JPEG (.jpg)
- In tali formati si utilizzano metodi di *compressione* per ridurre lo spazio di memorizzazione

Rappresentazione dei suoni

- Si effettuano dei campionamenti su dati analogici
- Si rappresentano i valori campionati con valori digitali
- La frequenza del campionamento determina la fedeltà della riproduzione del suono

Rappresentazione dati di interesse

- Interi = complemento a 2
- Razionali e reali = virgola mobile su 32 bit
- Puntatori = indirizzi = interi senza segno su 32 bit
- Stringhe = 8 bit per carattere

Distinzione di tipi

- Data una certa stringa di bit come si può distinguere se
  - Codifica intero senza segno
  - Codifica intero relativo
  - Codifica immagine
  - …
- Ci sono due possibili soluzioni
  - Associo alla stringa un'altra stringa (descrittore o metadato) che mi dice quale è il tipo della stringa
  - In base alle operazioni che applico, cioè applicata una certa operazione essa assume un certo tipo
- Il significato di un una stringa di bit dipende quindi dalle operazioni applicate alla stringa stessa