Università degli Studi di Cagliari
Corso di Laurea in Ingegneria Elettronica

ELEMENTI DI INFORMATICA

http://agile.diee.unica.it

A.A. 2015/2016

Docente: **Michele Marchesi**

CODIFICA BINARIA DELL'INFORMAZIONE
PARTE II

Numeri reali

- Un calcolatore in realtà rappresenta solo numeri razionali con parte intera e parte frazionaria.
- Due rappresentazioni possibili
  - **Virgola fissa *(fixed point)***: giustapposizione di parte intera e parte frazionaria separate dal punto decimale. Il numero di cifre per la parte intera e frazionaria è fissata a priori dalla posizione del punto decimale.
  - **Virgola mobile** ***(floating point)*** utilizza la notazione esponenziale

    $$r = m \cdot b^n$$

    r è il numero da rappresentare, ***m*** è un numero frazionario detto *mantissa*, ***b*** è una base e ***n*** è un intero con segno (*caratteristica*)

Virgola fissa

- Dato un numero N da rappresentare con n bit, si fissa quanti bit sono dedicati alla parte intera, i, e quanti alla parte frazionaria, f, $n = i + f$
- In pratica, si fissa la posizione che la virgola deve avere all'interno del numero da rappresentare
- Ad esempio: $N = 83.7$, $i = 8$, $f = 4$ ($n = 12$ bit)
- Parte intera: 83 = 01010011
  (ottenuto con le divisioni successive)
- Parte frazionaria: $0.7 \approx 0.1011$
  (ottenuto con le moltiplicazioni success.)
- N = 01010011.1011

Virgola fissa

- In pratica, si rappresenta il numero intero $N' = N\ 2^f$:
  - si moltiplica il numero N per 2^f in modo da aver un numero intero da rappresentare con n bit
  - si rappresenta tale numero, in complemento a 2, quindi si rappresentano anche i numeri negativi
  - i primi i bit del numero sono la parte intera (ottenibile con uno scorrimento a destra di f bit)
  - gli ultimi f bit rappresentano la parte frazionaria

Virgola fissa

- Rappresentazione “facile”, che usa appieno la rappresentazione dei numeri interi
- Tutti i numeri rappresentabili sono punti discreti equispaziati sull'asse reale
- Non rappresenta bene numeri frazionari molto grandi
- Non rappresenta bene numeri (frazioni) molto piccoli
- Esempi (decimali a 16 cifre):

  60000000000.00002

  00000000000.00007

Rappresentazione dei numeri reali in virgola mobile

- Ricordiamo: $r = m \cdot b^n$
- ***m*** è un numero frazionario detto *mantissa*, ***b*** è una base e ***n*** è un intero con segno (*caratteristica*)
- La caratteristica *n* rappresenta l'ordine di grandezza
- Possiamo rappresentare con precisione sia numeri molto grandi che molto piccoli.
- Nella rappresentazione binaria, $b = 2$
- Data una parola di *n* bit, se ne assegnano uno al segno, n_m alla mantissa (numero frazionario) e n_c alla caratteristica

Rappresentazione dei numeri reali in virgola mobile

- Possiamo rappresentare con precisione sia numeri molto grandi che molto piccoli.
- Mantissa normalizzata: numero frazionario tale che la cifra subito dopo il punto decimale è diversa da zero
- In tal modo, si rappresentano sempre i numeri con la massima precisione possibile
- Esempio:
- $(24.828125)_{10}$ ⇨ 11000.110101 (virgola fissa)

  0.11000110101 *mantissa*

  101 *caratteristica in base 2*

Mantissa Normalizzata

- Numeri decimali:

  $1234.56 = 0.123456\ 10^{4}$ $m = 0.123456$

  $0.004567 = 0.4567\ 10^{-2}$. $m = 0.4567$

- Numeri binari

  $1101.001 = 0.1101001\ 2^{4}$ $m = 0.1101001$

  $0.0001101 = 0.1101\ 2^{-3}$. $m = 0.1101$

- Numeri binari (standard IEEE): la mantissa è del tipo "1.*x*" e non "0.*x*"

  $1101.001 = 1.101001\ 2^{3}$ $m = 1.101001$

  $0.0001101 = 1.101\ 2^{-4}$. $m = 1.101$

Rappresentazione dei numeri reali in virgola mobile (cont.)

- Vantaggi della notazione in virgola mobile:
  - Numeri molto grandi vengono rappresentati con poche cifre
  - Numeri molto piccoli vengono rappresentati con precisione
- "Svantaggi":
  - La "distanza" fra due numeri reali consecutivi rappresentabili in virgola mobile è variabile:
    - *maggiore* fra numeri "grandi"
    - *minore* fra numeri "piccoli"
- Nel corso di **Calcolo Numerico** si valutano gli effetti degli errori di approssimazione dovuti ai calcoli su numeri reali eseguiti con l'aritmetica del calcolatore

Il formato IEEE 754

- L'IEEE ha standardizzato la rappresentazione in virgola mobile, nel 1985 e 2008 (standard IEEE 754)
- Prima dello standard, ogni produttore aveva la propria rappresentazione, con variazioni di n_m e n_c
- Ciò comportava una non completa equivalenza dei risultati dello stesso programma su varie macchine, dovuta ai differenti arrotondamenti
- Rappresentazioni a 16, 32, 64 e 128 bit: precisione dimezzata, semplice, doppia e quadrupla precisione

Il formato IEEE 754

- Lo standard definisce non solo numeri, ma anche valori speciali:

  NaN Not a Number, numero non definito

  +∞ numero maggiore del massimo rappresentabile

  -∞ numero minore del minimo rappresentabile

  +0 lo zero è rappresentato con tutti zeri

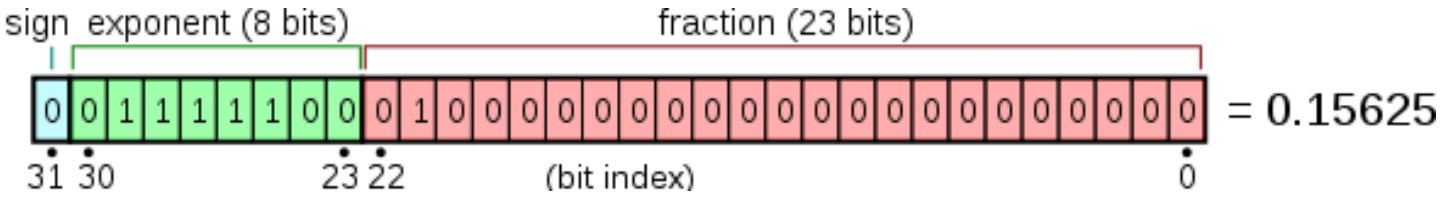
  -0 numero < 0 maggiore del massimo rappresent.

- Esso fissa anche:
  - Formati di codifica per scambio dati
  - Regole di arrotondamento
  - Operazioni aritmetiche
  - Gestione delle eccezioni (come divisione per zero, ecc.)

Il formato IEEE 754 a 32 bit

- Campi della parola a 32 bit:
  - s bit del segno (0 = +, 1 = -)
  - E caratteristica o esponente (8 bit) $E \in [1, 254]$
  - M mantissa (24 bit, di cui 23 memorizzati)
- Se $r = (-1)^s \cdot m \cdot 2^n$, allora $E = n + 127$, $n \in [-126,127]$
- Poiché nei numeri binari la mantissa normalizzata inizia sempre con "1.", questo "1" è omesso, guadagnando un bit:
  $m = 1 + M$

$E = 124 \rightarrow n = -3$; $M = 0.25 \rightarrow m = 1.25$; r = $1.25\ 2^{-3} = 0.15625$:



Il formato IEEE 754 a 32 bit

- Valori speciali:

  $E = 255, M \neq 0$: $r = \text{NaN}$

  $E = 255, M = 0, s = 0$: $r = +\infty$

  $E = 255, M = 0, s = 1$: $r = -\infty$

  $E = 0, M = 0, s = 0$: $r = +0$

  $E = 0, M = 0, s = 1$: $r = -0$

- Valori numerici:

  se $0 < E < 255$: $r = (-1)^s \cdot 2^{E-127} \cdot 1.M$

  se $E = 0$ e $M \neq 0$: $r = (-1)^s \cdot 2^{-126} \cdot 0.M$

Il formato IEEE 754: precisione

Name	Common name	Base	Digits	E min	E max	Notes	Decimal digits	Decimal E max
binary16	Half precision	2	10+1	−14	+15	storage, not basic	3.31	4.51
binary32	Single precision	2	23+1	−126	+127		7.22	38.23
binary64	Double precision	2	52+1	−1022	+1023		15.95	307.95
binary128	Quadruple precision	2	112+1	−16382	+16383		34.02	4931.77

Esercizio 1

- Convertire il numero 38.0625_{10} in formato IEEE 754 a 16 bit (1 bit segno, 5 per esponente, 10 per mantissa). L'esponente è codificato sommando 15 al valore vero.
- Formato binario:
  - parte intera: 38 = 32 + 4 + 2 = 100110
  - parte decimale. 0.0625 = 0.0001 →

    $38.062510 = 100110.0001 = 1.001100001\ 2^5$

  → la mantissa è quindi: 001100001 (l'uno a sinistra non è esplicitamente codificato);
- la caratteristica è 5, e si codifica: 5 + 15 =20 = 10100
- La codifica completa è: 0 10100 0011000010

Esercizio 2

- Convertire in decimale il numero IEEE 754 a 16 bit (1 bit per il segno, 5 per l'esponente, 10 per la mantissa):

  0100111010100000

- Il bit segno vale 0, quindi il numero è positivo.
- L'esponente è codificato sommando 15 al valore vero.
- L'esponente vale 10011, cioè 1 + 2 + 16 = 19; il valore vero è quindi 19 – 15 = 4.
- La mantissa vale: $1.1010100000 = 2^{-10}\ 11010100000 = = 2^{-10}\ (32+128+512+1024) = 2^{-10}\ 1696$
- Il numero è quindi: $1696\ 2^{-10}\ 2^{4} = 1696\ 2^{-6} = = 1696 / 64 = 26.5$

Esercizio 3

- Convertire in decimale il numero IEEE 754 a 16 bit (1 bit per il segno, 5 per l'esponente, 10 per la mantissa):

  1101010101010101

- Il bit segno vale 1, quindi il numero è negativo.
- L'esponente è codificato sommando 15 al valore vero.
- L'esponente vale 10101, cioè 1 + 4 + 16 = 21; il valore vero è quindi 21 – 15 = 6.
- La mantissa vale: $1.0101010101 = 2^{-10}\ 10101010101 =$
  $= 2^{-10}\ (1+4+16+64+256+1024) = 2^{-10}\ 1365$
- Il numero è quindi: $-1365\ 2^{-10}\ 2^{6} = -1365\ 2^{-4} =$
  $= -1365 / 16 = -85.3125$

Riferimenti bibliografici

- Per la rappresentazione dei numeri e le conversioni si possono consultare i siti di Wikipedia (meglio se in inglese):
  - Numeri binari: en.wikipedia.org/wiki/Binary_numeral_system
  - Complemento a 1: en.wikipedia.org/wiki/Ones%27_complement
  - Complemento a 2: en.wikipedia.org/wiki/Two%27s_complement
  - Virgola fissa: en.wikipedia.org/wiki/Fixed-point_arithmetic
  - Virgola mobile: en.wikipedia.org/wiki/Floating_point
  - IEEE 754: en.wikipedia.org/wiki/IEEE_754
- Esercizi (in italiano) si trovano facilmente su Web, ricercando ad es.: “conversione decimale binario esercizi”, “formato IEEE 754 esercizi” o simili

Codifica dei caratteri

- I caratteri vengono codificati tramite sequenze di bit.
- Codice più usato: **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) a 7 bit (128 caratteri), di solito esteso a 8 bit (256 caratteri)
- Tre categorie di caratteri
  - **Caratteri di comando**: codici di trasmissione o di controllo
  - **Caratteri alfanumerici**: da ‘A’ a ‘Z’, da ‘a’ a ‘z’ e da ‘0’ a ‘9’
  - **Simboli**: punteggiatura e operatori aritmetici
  - Le accentate, i caratteri greci ecc. fanno parte del codice esteso.
  - Estensione ISO a 16 bit permette di rappresentare tutti gli alfabeti (arabo, cirillico, giapponese, indiano, ecc.)

Codifica ASCII: car. di comando

Binary	Oct	Dec	Hex	Abbr	PR[a]	CS[b]	CEC[c]	Description
000 0000	000	0	00	NUL	NUL	^@	\0	Null character
000 0001	001	1	01	SOH	SOH	^A		Start of Header
000 0010	002	2	02	STX	STX	^B		Start of Text
000 0011	003	3	03	ETX	ETX	^C		End of Text
000 0100	004	4	04	EOT	EOT	^D		End of Transmission
000 0101	005	5	05	ENQ	ENQ	^E		Enquiry
000 0110	006	6	06	ACK	ACK	^F		Acknowledgment
000 0111	007	7	07	BEL	BEL	^G	\a	Bell
000 1000	010	8	08	BS	BS	^H	\b	Backspace[d][i]
000 1001	011	9	09	HT	HT	^I	\t	Horizontal Tab
000 1010	012	10	0A	LF	LF	^J	\n	Line feed
000 1011	013	11	0B	VT	VT	^K	\v	Vertical Tab
000 1100	014	12	0C	FF	FF	^L	\f	Form feed
000 1101	015	13	0D	CR	CR	^M	\r	Carriage return[h]
000 1110	016	14	0E	SO	SO	^N		Shift Out
000 1111	017	15	0F	SI	SI	^O		Shift In

Codifica ASCII: car. di comando

Binary	Oct	Dec	Hex	Abbr	PR[a]	CS[b]	CEC[c]	Description
001 0000	020	16	10	DLE	DLE	^P		Data Link Escape
001 0001	021	17	11	DC1	DC1	^Q		Device Control 1 (oft. XON)
001 0010	022	18	12	DC2	DC2	^R		Device Control 2
001 0011	023	19	13	DC3	DC3	^S		Device Control 3 (oft. XOFF)
001 0100	024	20	14	DC4	DC4	^T		Device Control 4
001 0101	025	21	15	NAK	NAK	^U		Negative Acknowledgement
001 0110	026	22	16	SYN	SYN	^V		Synchronous Idle
001 0111	027	23	17	ETB	ETB	^W		End of Trans. Block
001 1000	030	24	18	CAN	CAN	^X		Cancel
001 1001	031	25	19	EM	EM	^Y		End of Medium
001 1010	032	26	1A	SUB	SUB	^Z		Substitute
001 1011	033	27	1B	ESC	ESC	^[	\e[f]	Escape[g]
001 1100	034	28	1C	FS	FS	^\		File Separator
001 1101	035	29	1D	GS	GS	^]		Group Separator
001 1110	036	30	1E	RS	RS	^^		Record Separator
001 1111	037	31	1F	US	US	^_		Unit Separator
111 1111	177	127	7F	DEL	DEL	^?		Delete[e][i]

Codifica ASCII: alfanumerici e simboli

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p

Codifica ASCII: alfanumerici e simboli

Binary	Oct	Dec	Hex	Glyph
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Aritmetica dei caratteri

- I caratteri ASCII sono codificati entro un un byte
- I caratteri alfanumerici hanno codifiche consecutive:
  - Cifre: `'0'-> 48, '1'-> 49, …, '9'-> 57`.
  - Maiuscole: `'A'-> 65, 'B'-> 66, …, 'Z'-> 90`.
  - Minuscole: `'a'-> 97, 'b'-> 98, …, 'z'-> 122`.
- In linguaggio C, il tipo `char` equivale a un intero positivo di 8 bit
- Sono ammesse operazioni aritmetiche sui caratteri (attenti all'overflow) e assegnazioni di caratteri a interi e viceversa

Aritmetica dei caratteri

- Perché usare caratteri come interi:
  - Per codificare interi piccoli con un solo byte
  - Per operazioni di conversione intero-carattere
- Assegnando a un carattere un intero *i* > 255, viene assegnato in realtà: *i* mod 256 (il resto della divisione per 256)
- Esempio: conversione da intero tra 0 e 9 alla cifra corrispond.:

```
int i = 7;
char c = i + 48;  // c contiene ora '7'
```

- O meglio:

```
int i = 7;
char c = i + '0';
```

- Esempio: overflow

```
int i = 256;
char c = i;  // c contiene 0
```

Aritmetica dei caratteri

- Conversione da cifra a numero:

```
char c = '5';
int i = c – '0';   // ora i vale 5
```

- Confronto tra caratteri (ordinamento alfabetico)
- Tra i numeri, valgono gli **operatori di confronto**:

```
==   >   <   >=   <=   !=
```

- Gli stessi valgono anche per i caratteri, quindi col confronto si può accertare quale di due caratteri alfabetici maiuscoli o minuscoli viene prima in ordine alfabetico:
  - `'Z' > 'X'    // vero!`
  - `'a' < 'B'    // falso! (non si possono // confrontare maiuscole e minuscole`

Codifica delle immagini

- L'immagine è suddivisa in punti (pixel) e ciascun punto è codificato con un numero che corrisponde
  - A un particolare colore
  - A un particolare tono di grigio nelle immagini b/n
- In genere si utilizza un numero di colori o di sfumature di grigio che sia potenza di 2 per rappresentare un'immagine come sequenza di byte.
- Deve essere memorizzata anche la dimensione dell'immagine e la risoluzione (dpi, "*dot per inch*")
- La quantità di informazione può essere rilevante. Es. 2000 x 2000 px, 2 x 3 Byte → 24 MByte

Codifica delle immagini

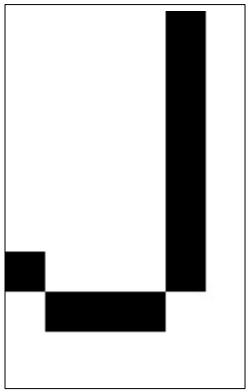
- Nelle immagini a colori, viene memorizzato il livello di intensità dei colori fondamentali
- Nel modello RGB i colori sono: rosso, verde e blu
- In CMYK, usato per la stampa, essi sono: ciano, magenta, giallo e nero
- Il numero di colori o di livelli di grigio possibili dipende dai bit utilizzati:
  - un'immagine con 1 bit per pixel avrà al massimo due valori possibili (0 e 1), quindi solo bianco e nero
  - con 8 bit per pixel, si possono rappresentare al massimo 256 colori o 256 livelli di grigio
  - Con 16 bit per pixel i colori/livelli sono 65536

File grafici

- Vi sono molti modi per memorizzare un'immagine in un file (formati diversi di file grafici)
- Formati **compressi** e no
- I formati compressi possono essere:
  - **Lossless**: non si perde informazione, ideale per disegni tecnici e immagini “geometriche”
  - **Lossy**: l'immagine decompressa non è identica a quella originale, va bene per fotografie
- Tutti i formati hanno un'intestazione (header) che contiene le informazioni per interpretare l'immagine, che segue

Formati non compressi

- BMP: formato di Windows
- PBM (Portable Bit Map), PGM (Portable Gray Map), PPM (Portable Pixel Map) formati ASCII molto semplici:
  - PBM per pixel bianchi o neri, PGM per pixel con scala di grigio, PPM per pixel a colori RGB
- TIFF (o TIF): formato che può essere sia compresso che non compresso
- Esempio di PBM:



```
P1
# Esempio con la lettera "J"
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

Formati compressi lossless

- PNG (Portable Network Graphics): formato aperto con 16M di colori, ideale per Web browser e molto diffuso. Possibili anche animazioni.
- GIF (Graphic Interchange Format): formato proprietario (ma in pratica molto diffuso) con 256 colori. Permette anche animazioni (gruppi di immagini visualizzate una dopo l'altra, con una data temporizzazione)
- TIFF, TIF: possibile compressione con o senza perdite. Usato per applicazioni professionali.

Formato “lossy” JPEG

- JPEG (Joint Photographic Experts Group): formato standard molto usato per immagini fotografiche
- Compressione dell'immagine digitale a tono continuo, sia a livelli di grigio che a colori.
- È un formato aperto e gratuito
- L'immagine decompressa non è identica all'originale, ma tende a perdere dettagli poco percepibili dall'occhio umano
- Possibili diversi livelli di compressione, e quindi di errore dell'immagine decompressa
- Immagine con qualita della compressione che decresce da sinistra a destra:

Codifica dei suoni

- I suoni udibili (variazione della pressione dell'aria tra circa 20 Hz e 20 Khz) sono codificati in forma digitale (seq. di numeri), su 1 o 2 canali (stereo)
- Formato standard di codifica: PCM (Pulse-code modulation), caratterizzato da
  - Frequenza di campionamento
  - Profondità di bit: nr. di bit per codificare i campioni
- Formati file (PCM + header):
  - .WAV (Windows), .AIFF (Mac OS)
  - .BWF: standard europeo professionale che include *metadati*

File sonori

- Poiché i file sonori tendono a essere piuttosto grandi (ca. 10 Mb/min per audio di buona qualità), essi sono tipicamente **compressi**
- I formati compressi possono essere **Lossless**: non si perde informazione (FLAC, WavPack, Monkey's Audio, ALAC); però non si guadagna molto (circa un fattore 2)
- **Lossy**: i più usati: si perde qualcosa, ma la compressione può guadagnare anche un fattore 10

Il formato MP3

- MPEG-1 or MPEG-2 Audio Layer III (da non confondere con MPEG-3), standard ISO 1993
- E' il più comune formato per comprimere file audio
- Si basa su principi di *psicoacustica*: nella compressione sono perse per lo più informazioni poco udibili dall'orecchio umano
- I dispositivi di riproduzione del suono sono in grado di riprodurlo direttamente, senza prima decomprimerlo in un file PCM
- La compressione ottenuta è circa 1/11 del file PCM

Codifica dei video

- I video sono codificati in forma digitale come sequenza di immagini
- Sono tipicamente associati all'audio corrispondente, compresso e sincronizzato
- Viste le grandi dimensioni, sono tipicamente codificati in modo compresso e **lossy**
- Poiché un video è una sequenza di immagini molto simili (tranne che nei cambi di scena), la codifica registra solo le differenze tra un fotogramma e il successivo
- Standard più usati: ISO Moving Picture Experts Group MPEG-1, -2 e -4