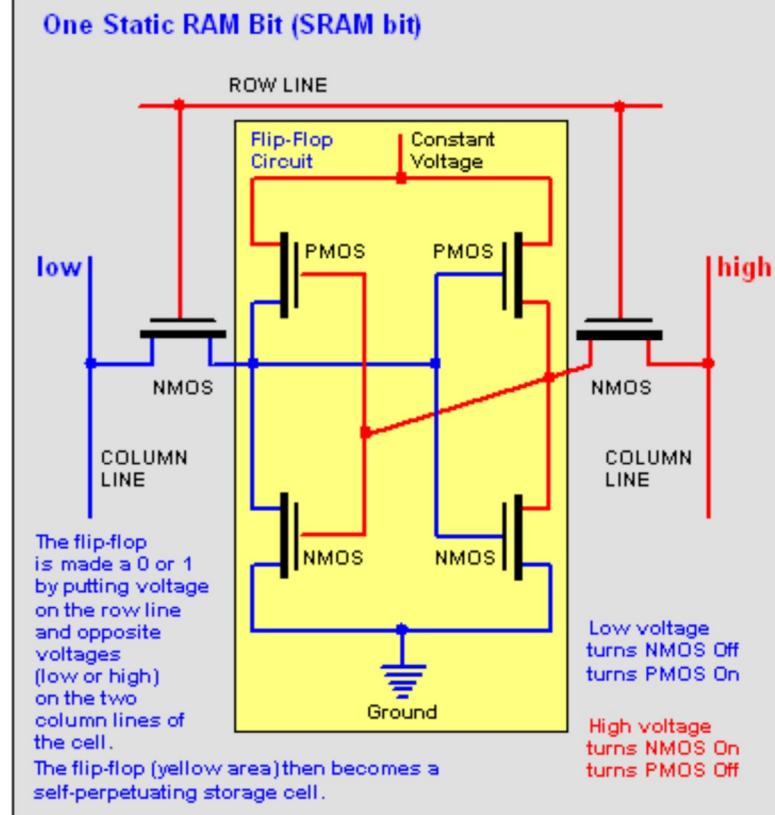
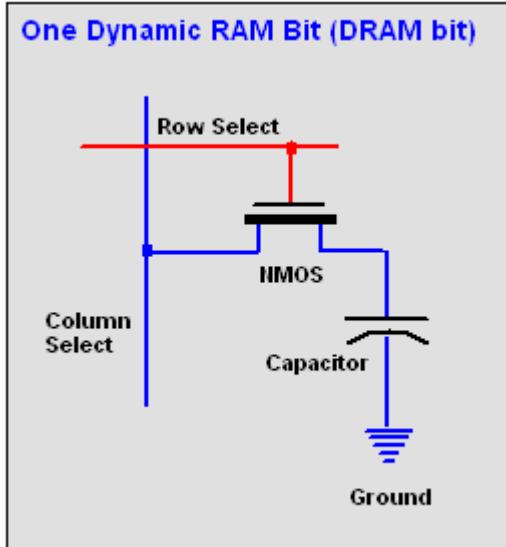




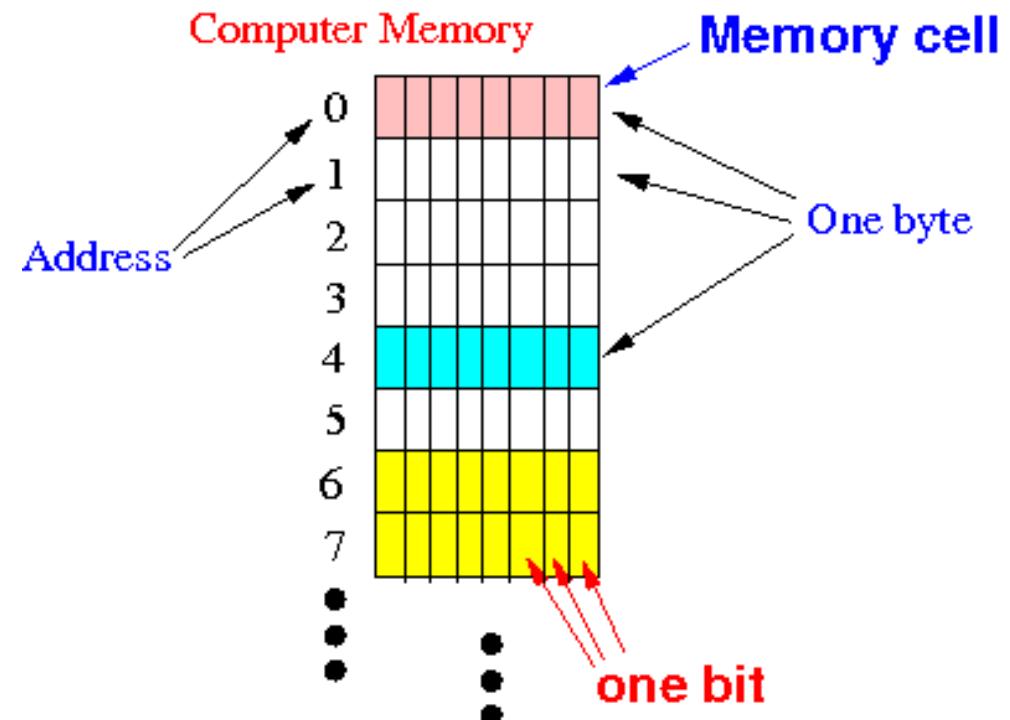


- Riepilogo memoria
- Rappresentazione e numero
- Rappresentazione decimale
- Rappresentazione binaria
- Rappresentazione esadecimale
- Numeri negativi
- Numeri non interi
- Simboli





- Dati e codice → memoria
- Come li memorizzo?
- Elementi a disposizione
  - Byte composti da
  - bit





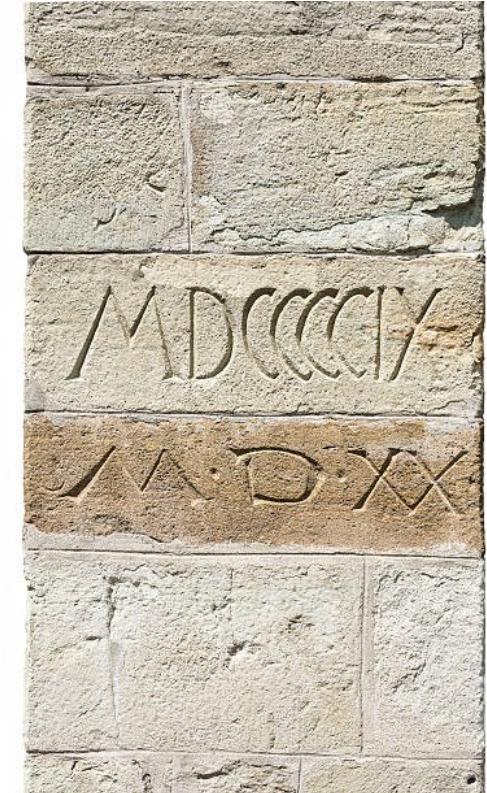
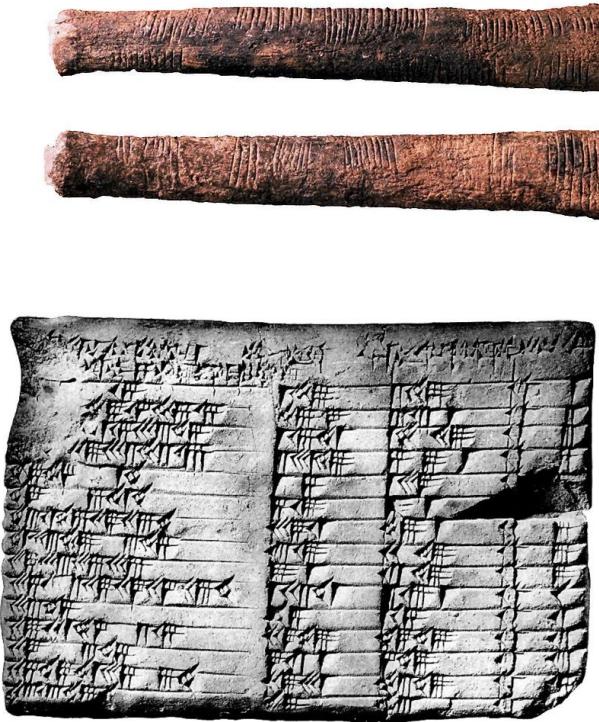
- L'elemento base di memoria è il bit
- Assume formalmente i valori di 0 e 1
- Tutto ciò che può venir memorizzato in memoria deve quindi essere combinazione di due simboli (1 e 0)
  - Vale anche per altri tipi di memoria
- **Impatto su sistemi di rappresentazione dei numeri in informatica**

THE PHONETIC ALPHABET MORSE CODE GUIDE		
A ALPHA	--	N NOVEMBER --
B BRAVO	----	O OSCAR -----
C CHARLIE	---	P PAPA -----
D DELTA	---	Q QUEBEC -----
E ECHO	.	R ROMEO ---
F FOXTROT	----	S SIERRA ---
G GOLF	---	T TANGO -
H HOTEL	----	U UNIFORM ---
I INDIA	..	V VICTOR ---
J JULIET	----	W WHISKEY ---
K KILO	--	X X-RAY -----
L LIMA	---	Y YANKEE -----
M MIKE	--	Z ZULU -----

# Rappresentazione e Numero



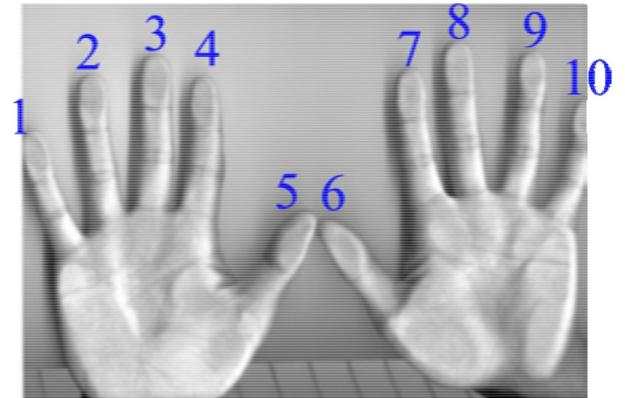
# Rappresentazione e numero



# Rappresentazione Decimale



- Basato su 10 simboli (**base 10**):
  - 0, 1 ,2 ,3 ,4 ,5, 6, 7, 8, 9
- Sistema posizionale
  - La posizione di ciascun simbolo è importante
    - 1786 non è 8761
- Non sono però regole assolute
  - MMXXII
  - Quatre-vingt



- Per trascrivere i numeri qualcuno ad un certo punto inventò dei simboli
  - Ma i numeri sono tanti...
- Se iniziamo a contare:
  - 0 1 2 3 4 5 6 7 8 9 ... e poi?
  - Si aggiunge una seconda colonna con un altro significato
  - 90

- Come calcolo il valore di un numero scritto in **base 10**?
- Date le possibili cifre d:
  - $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Un numero di N cifre è rappresentabile come:
  - “ $d_{n-1}d_{n-2}\dots d_2d_1d_0$ ”
  - Esempio: 1786
- Il suo valore V è calcolabile come:
  - $V = d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + \dots + d_2 \times 10^2 + d_1 \times 10 + d_0$
  - Ovvero come somma di potenze del 10
  - Esempio:  $1786 = 1 \times 1000 + 7 \times 100 + 8 \times 10 + 6$

- Nelle celle di memoria solo due stati → due simboli (1 e 0)
- Uso sistema posizionale a base 2 → sistema binario
- Cifre:
  - $b \in \{0, 1\}$
- Rappresentazione:
  - $b_{n-1}b_{n-2}\dots b_2b_1b_0$
- Valore
  - $V = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_2 \times 2^2 + b_1 \times 2 + b_0$

# Esempio

- Il numero binario rappresentato come 101110 che valore ha?
- Somma di potenze di due
  - $V = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 0$
  - $V = 32 + 8 + 4 + 2$

0	0	1	0	1	1	1	0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
<b>128</b>	<b>64</b>	<b>32</b>	<b>16</b>	<b>8</b>	<b>4</b>	<b>2</b>	<b>1</b>

- 1) Posizionarsi sulla sinistra del numero da convertire (parte più significativa)
- 2) Inizializzare N a 0
- 3) Leggere la cifra binaria in esame
- 4) Moltiplicare N per 2 e sommarvi la cifra letta
- 5) Se ci sono ancora cifre procedere verso destra e tornare al punto 3)
- 6) Fine della procedura, N contiene il risultato

- 1) Inizializzare N come il numero da convertire
- 2) Inizializzare una sequenza di caratteri S come vuota
- 3) Si divide N per 2
- 4) Il resto è la cifra meno significativa ancora da calcolare, accumularlo in S verso destra
- 5) Il risultato è 0?

Sì: fine

No: porre N = risultato e ripetere da 3

- Altri sistemi usati sono:
  - Rappresentazione esadecimale, base 16:
    - Cifre in 0-9 e A-F
    - In pratica 1 cifra codifica 4 bit
    - Esempio: 0x90FF
  - Rappresentazione ottale, base 8:
    - Cifre in 0-7
    - 1 cifra può essere codificata con 3 bit
- In entrambi i casi la conversione da e verso il binario è semplice

- Conversione da binario ad esadecimale: si prendono i bit 4 a 4 (partendo dai meno significativi) e si sostituiscono con la corrispondente cifra esadecimale:
  - $1011\ 1001_2 = 0xB9_{16}$
- In pratica si usa tabella di conversione

# Tabella di Conversione

Decimale	Binario	Esadecimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



- Corto rispetto binario ma sempre potenze di 2
  - 11011111010 → 0x6FA → 1786
  - $16 = 2^4$
- Indirizzi di memoria
- Valore byte raw
- Codifica colori RGB
  - 3 byte
  - 0xA13A77

- Valgono le stesse regole dell'aritmetica in base 10!
- Per qualunque sistema posizionale in qualunque base

$$\begin{array}{r}
 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 \end{array}$$

Bit 0 :

$$\begin{array}{r}
 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 & & & & 1
 \end{array}$$

Bit 1 :

$$\begin{array}{r}
 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 & & & & 0 & 1
 \end{array}$$

Bit 2 :

$$\begin{array}{r}
 \text{Carry } 1 & 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 & 0 & 0 & 1
 \end{array}
 \quad 1+1=2=10$$

Bit 3 :

$$\begin{array}{r}
 \text{Carry } 1 & 1 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 & 0 & 0 & 1
 \end{array}
 \quad 1+1=2=10$$

Bit 4 :

$$\begin{array}{r}
 \text{Carry } 1 & 1 & 1 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 & 0 & 0 & 1
 \end{array}
 \quad 1+1+1=3=11$$

Result

$$\begin{array}{r}
 1 & 1 \\
 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 \\
 \hline
 = & 1 & 1 & 0 & 0 & 1
 \end{array}$$

- La memoria è organizzata in bit MA
- Questi vengono organizzati in “parole”
  - Tipicamente multipli di 8 byte
- Quindi:
  - Per rappresentare numeri obbligo  $n \times 8$  cifre
  - In generale numero cifre limitato

- Una parola binaria di  $n$  cifre (bit) permette di rappresentare tutti i numeri interi  $N$  tali che:
  - $0 \leq N \leq 2^n - 1$
- Dato un numero  $N$  la sua rappresentazione binaria richiedera un numero  $n$  di bit tale che:
  - $n > \log_2 N$

- Esempi:
  - 1 byte ovvero 8 cifre binarie mi permettono di memorizzare i numeri tra 0 e 255
  - 256 valori differenti
- Il numero decimale  $17899_{10}$  richiederà almeno 15 cifre binarie ( $\log_2(17899)=14,12759..$ )
  - Ma noi dovremo -probabilmente- usare 2 byte

# Ci bastano i numeri naturali?



- No, non ci bastano.
- Dovremo affrontare:
  - Numeri negativi
  - Numeri non interi



- Devono:
  - mantenere la stessa rappresentazione per i numeri positivi
  - utilizzare rappresentazioni alternative per i numeri negativi
- Inoltre:
  - semplicità di utilizzo
  - semplicità di implementazione
- $N$  bit  $\rightarrow 2^N$  possibili combinazioni
  - $2^N$  possibili numeri

# Bit di Segno



- A un bit associo il significato di segno
  - $0 \rightarrow +$
  - $1 \rightarrow -$
- Molto intuitivo
- Tuttavia bit che va trattato diversamente
- Complicazioni
  - SW
  - HW

Binario	Decimale
0 000	0
0 001	1
0 010	2
0 011	3
0 100	4
0 101	5
0 110	6
0 111	7
1 000	-0
1 001	-1
1 010	-2
1 011	-3
1 100	-4
1 101	-5
1 110	-6
1 111	-7

# Complemento a 1

- Nessun bit speciale
- Numeri positivi come visto
- Numeri negativi → complemento
  - 1 diventa 0 e viceversa
- 8 bit → 255 valori
  - [-127,127]
  - Lo 0 è rappresentato in due modi

Binario	Decimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

# Complemento a 2

- Sistema di fatto usato nelle architetture moderne
- Numeri positivi
  - Rappresentazione vista fino ad adesso
- Per ottenere negativo
  - Complemento
  - Aggiungo 1
- Per ottenere positivo da negativo
  - Stessa operazione!

Binario	Decimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

- Intervallo di valori
  - $[-2^{N-1}, 2^{N-1}-1]$
  - Esempio 8 bit  $\rightarrow [-128, 127]$
- Vantaggi
  - Non ho +0 e -0
  - Sfrutto meglio dinamica possibile
  - Il bit più significativo mi indica comunque il segno senza essere bit di segno
  - Stesse operazioni aritmetiche

- Conversione rapida
  - si lasciano inalterati gli 0 a partire dal bit meno significativo
  - si lascia inalterato il primo 1
  - si complementa tutto il resto

0110111000 → 1001001000

# Ci bastano i numeri naturali?



- No, non ci bastano.
- Dovremo affrontare:
  - Numeri negativi
  - Numeri non interi



- Come facciamo con il sistema decimale?
  - Esempio: 123,456
  - Lo vedo come  $1 \times 100 + 2 \times 10 + 3 + 4/10 + 5/100 + 6/1000$
  - I numeri dopo la “virgola” sono comunque potenze del 10
  - $V(123,456) = 1 \times 10^2 + 2 \times 10 + 3 + 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$
- Facilmente estendibile al sistema binario!
  - Esempio: 1011.1011
  - $V(1011.1011) = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2 + 1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$
  - $V(1011.1011) = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2 + 1 + 1 \times \frac{1}{2} + 0 \times (\frac{1}{2})^2 + 1 \times (\frac{1}{2})^3 + 1 \times (\frac{1}{2})^4$

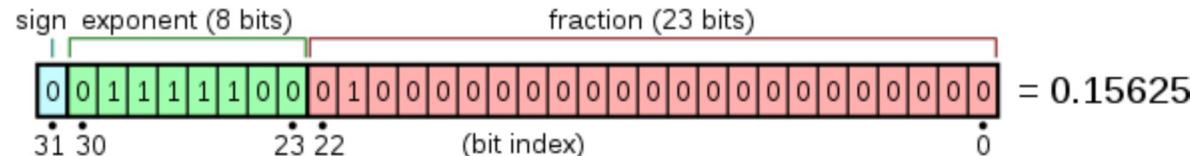
- Un numero che in formato decimale ha finite cifre non è detto abbia lo stesso proprietà formato binario
  - Esempio:  $0.3 \rightarrow 0.01\overline{0011}$
  - E viceversa
- Come vedremo ha un impatto quando gestiremo numeri non interi in C
  - Approssimazioni!

- Anche qui problema del numero “prefissato” di cifre
- Numeri a virgola fissa
  - Degli N byte a disposizione ne dedico M alla parte frazionaria
- Poco usato
  - Sistemi embedded

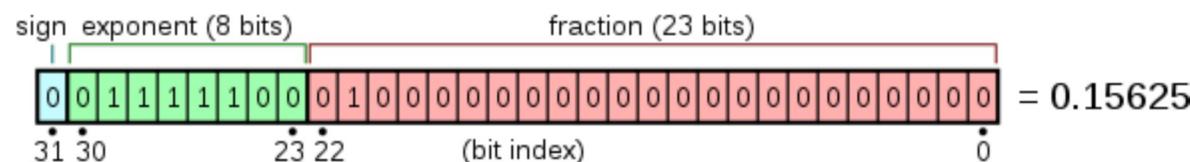
- Nel sistema decimale si usa anche la notazione “scientifica”
- $-1234,567 \rightarrow -1.234567 \times 10^3$
- Il formato generale lo posso vedere come
- (segno) (parte intera).(mantissa)  $\times 10^{(\text{esponente})}$ 
  - $0 < (\text{parte intera}) < 10$
- Facilmente estendibile al sistema binario

# Numeri a virgola mobile

- Standard IEEE 754 (Standard for Binary Floating-Point Arithmetic)
- Usa da 16 a 80 bit a seconda della precisione desiderata:
  - Parte intera sempre 1 (posso ometterla)
  - Bit di segno S
  - Esponente E
  - Mantissa M con  $\frac{1}{2} < M < 1$
  - $N = (-1)^s \cdot (1.M) \cdot 2^E$
- E ed M, assumono valori specifici (0,255) per gestione eccezioni (NaN,  $\pm\infty$ , underflow)



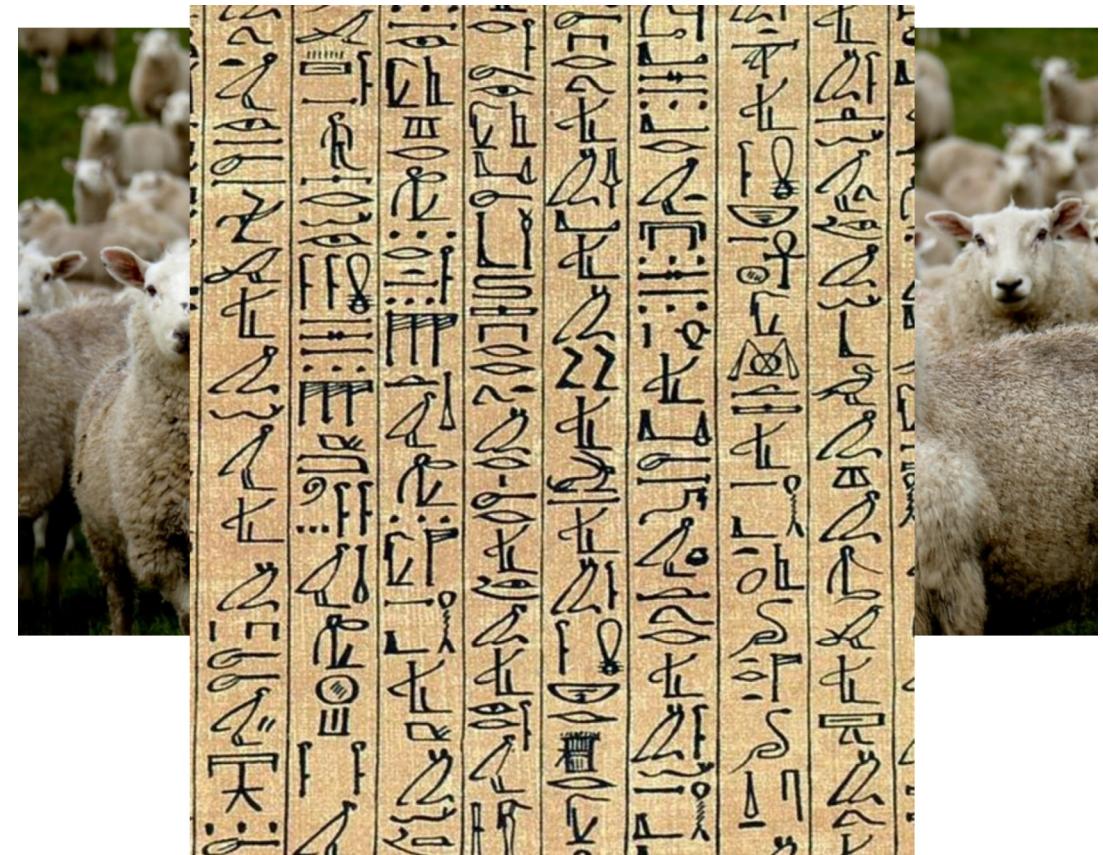
- Come lo calcolo?
  - Bit di segno → 0 → numero positivo
  - Esponente → 124
    - Ma viene sottratto un *bias* (127)
    - → -3
  - Mantissa → 0.25 ovvero  $(\frac{1}{2})^2$
  - Parte intera → 1
- $0^0 \times 1.25 \times 2^{-3} \rightarrow 0.15625$



# Ci bastano i numeri naturali?



- No, non ci bastano.
- Dovremo affrontare:
  - Numeri negativi
  - Numeri non interi
  - Simboli!



- Come memorizzo i simboli?
- Associo a ciascun simbolo un numero!
  - Tabella: Simbolo → Numero
- Alcuni standard
  - ASCII → American Standard Code for Information Interchange
  - EBCDIC → Extended Binary Coded Decimal Interchange Code
  - UTF-x → Unicode Transformation Format

- 7 bit
  - Costi di trasmissione/bit di parità
  - Numeri da 0 a 127
- 0-31 e 127 → caratteri di controllo
- 32-126 → caratteri stampabili
- 128-255 → Extended ASCII per caratteri “speciali”
  - In pratica estensioni nazionali

ASCII control characters			ASCII printable characters								Extended ASCII characters							
			32	space	64	@	96	'	128	ç	160	á	192	l	224	ó		
00	NULL	(Null character)	32	space	64	@	96	'	128	ç	160	á	192	l	224	ó		
01	SOH	(Start of Header)	33	!	65	#	97	-	129	đ	161	é	193	ł	225	þ		
02	STX	(Start of Text)	34	“	66	“	98	”	130	đ	162	í	194	ł	226	ô		
03	ETX	(End of Text)	35	”	67	”	99	„	131	đ	163	ó	195	ł	227	ò		
04	EOT	(End of Trans.)	36	£	68	£	100	„	132	đ	164	ñ	196	ł	228	ð		
05	ENQ	(Enquiry)	37	¤	69	¤	101	„	133	đ	165	œ	197	ł	229	ö		
06	ACK	(Acknowledgement)	38	¤	70	¤	102	„	134	đ	166	œ	198	ł	230	µ		
07	BEL	(Bell)	39	¤	71	¤	103	„	135	đ	167	œ	199	ł	231	þ		
08	BS	(Backspace)	40	¤	72	¤	104	„	136	đ	168	œ	200	ł	232	þ		
09	HT	(Horizontal Tab)	41	¤	73	¤	105	„	137	đ	169	œ	201	ł	233	ú		
10	LF	(Line feed)	42	¤	74	¤	106	„	138	đ	170	œ	202	ł	234	ú		
11	VT	(Vertical Tab)	43	¤	75	¤	107	„	139	đ	171	œ	203	ł	235	ù		
12	FF	(Form feed)	44	¤	76	¤	108	„	140	đ	172	œ	204	ł	236	ý		
13	CR	(Carriage return)	45	¤	77	¤	109	„	141	đ	173	œ	205	ł	237	ÿ		
14	SO	(Shift Out)	46	.	78	N	110	n	142	ä	174	«	206	±	238	-		
15	SI	(Shift In)	47	/	79	O	111	o	143	å	175	»	207	¤	239	,		
16	DLE	(Data link escape)	48	0	80	P	112	p	144	é	176	¤	208	ð	240	=		
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	¤	209	đ	241	±		
18	DC2	(Device control 2)	50	2	82	R	114	r	146	æ	178	¤	210	ê	242	=		
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	¤	211	ë	243	%		
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	¤	212	è	244	¶		
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	â	213	í	245	§		
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	â	214	í	246	÷		
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù	183	â	215	î	247	,		
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	ö	248	º		
25	EM	(End of medium)	57	9	89	Y	121	y	153	ö	185	‡	217	»	249	"		
26	SUB	(Substitute)	58	:	90	Z	122	z	154	ü	186		218	„	250	.		
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	】	219	„	251	¹		
28	FS	(File separator)	60	<	92	\	124		156	£	188	】	220	„	252	³		
29	GS	(Group separator)	61	=	93	]	125	}	157	ø	189	¢	221	„	253	²		
30	RS	(Record separator)	62	>	94	^	126	~	158	x	190	¥	222	„	254	■		
31	US	(Unit separator)	63	?	95	_			159	f	191	„	223	„	255	nnbsp		
127	DEL	(Delete)																

I PC in laboratorio hanno tastiera italiana

Come ottenere i simboli necessari?

ALT + 3 cifre codice ASCII  
Ad esempio ALT+123 → “{“

