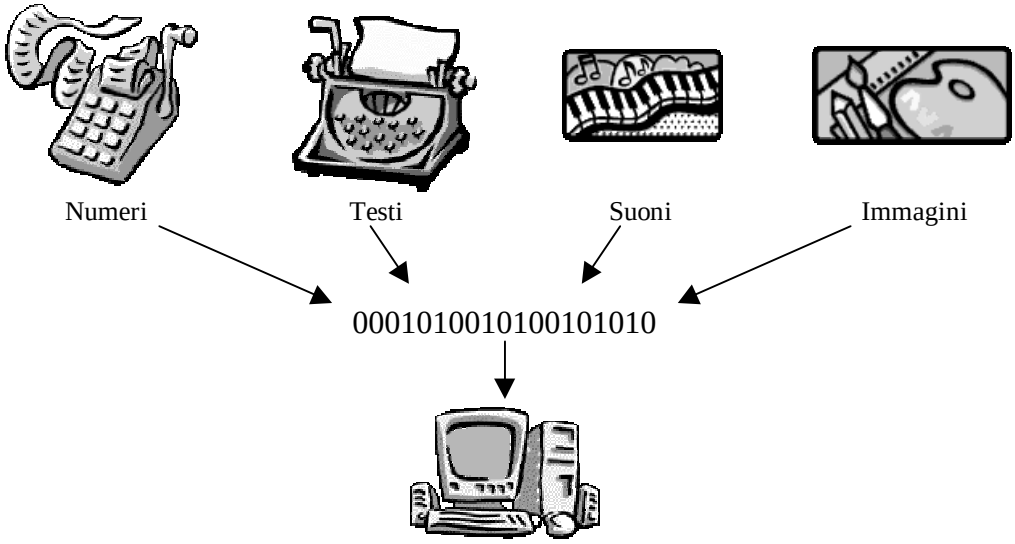
Informazione e computer

Si può rappresentare l'informazione attraverso varie forme:



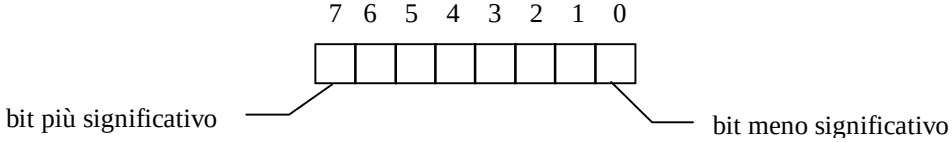
Computer

Cerchiamo di capire come tutte queste informazioni possano essere rappresentate all'interno di un elaboratore. Il calcolatore è in grado di manipolare solo simboli elementari che noi codifichiamo normalmente in "0" ed "1". Tutte le informazioni possono essere rappresentate mediante stringhe composte dai simboli "0" e "1".

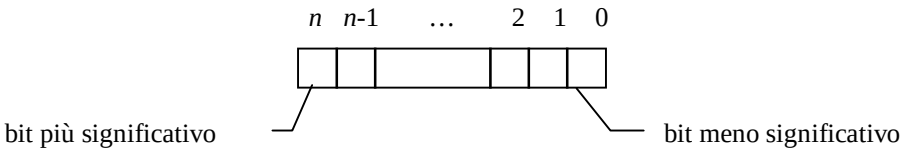
Notazioni e definizioni

- **BIT**: la parola BIT deriva da BInary digiT, termine inglese che significa cifra binaria e che noi siamo abituati a rappresentare con i simboli "0" e "1". Nel calcolatore non ci sono in ogni modo 0 ed 1, essendo una macchina elettrica l'informazione è rappresentata mediante due stati: livello di tensione "0 volt" e livello di tensione "5 volt". Lo stato zero indica una lampadina spenta mentre lo stato 1 una lampadina accesa.
- **BYTE**: è l'entità elementare di memorizzazione dell'informazione ed è costituita da 8 bit.

È facile comprendere che con 2 soli simboli è possibile rappresentare un'informazione estremamente limitata e, quindi, bisogna aggregare questi simboli in entità che ci consentono di rappresentare informazioni ben più complesse. È stata assunta un'entità standard costituita da 8 bit che costituiscono l'elemento di memorizzazione fondamentale. Possiamo pensare che un byte è costituito da una *sequenza ordinata* di 8 bit, dove il bit più a destra assume la posizione convenzionale 0 ed il bit più a sinistra assume la posizione convenzionale 7.



Più in generale se abbiamo $n+1$ bit partiremo dalla posizione 0, il cosiddetto bit meno significativo e raggiungeremo la posizione n-esima che rappresenta il bit più significativo.



Sono importanti pure i *multipli del byte*:

Kilobyte (Kb)	→	1.024 byte	(2^{10} byte)
Megabyte (Mb)	→	1.048.576 byte	(2^{20} byte)
Gigabyte (Gb)	→	1.073.741.824 byte	(2^{30} byte)

Come si riesce a codificare l'informazione all'interno di un computer?

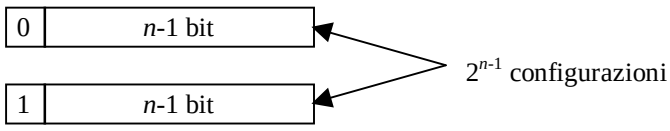
Noi stessi umani per la stessa entità usiamo forme diverse di rappresentazione:
se io volessi rappresentare l'informazione tavolo, in forma scritta potrei scrivere "tavolo" ma potrei anche dire "table" se dovessi parlare con un inglese, se mi rivolgessi ad un bambino farei un disegnino. Ho usato codifiche diverse per rappresentare lo stesso oggetto.
Come si può codificare all'interno di un elaboratore un testo scritto? Una pagina di un libro? Una lettera?

Noi siamo abituati a vedere un testo costituito da una serie di caratteri: i caratteri alfabetici tradizionali, le cifre del sistema decimale che utilizziamo, i caratteri di interpunzione.
Dobbiamo capire come tutti questi simboli possono essere rappresentati da codifiche di 0 ed 1. Il nostro problema è dunque quello di rappresentare *n* simboli (i caratteri che costituiscono la lettera) con un certo numero di bit.

Numero di oggetti rappresentabili con *n* bit
Con *n* bit è possibile rappresentare 2^n simboli.
Per dimostrarlo utilizziamo il *metodo induttivo*.

Con 1 bit è possibile rappresentare due simboli: uno che corrisponde allo 0 e l'altro all'1.
Con 2 bit, 4 simboli 00, 01, 10, 11.
Con 3 bit, 8 simboli 000, 001, 010, 011, 100,101, 110, 111.
Con n - 1 bit si possono rappresentare 2^{n-1} simboli. E con n bit?
Usare n bit significa aggiungere un bit ai n-1. Tale bit può avere solo due configurazioni: 0 o 1. Quindi con n bit avremo 2^{n-1} simboli che corrispondono al valore 0 del bit aggiunto (l'n-esimo) e 2^{n-1} che corrispondono al valore 1 del bit aggiunto, quindi:

$$2^{n-1} + 2^{n-1} = (1 + 1) * 2^{n-1} = 2 * 2^{n-1} = 2^1 * 2^{n-1} = 2^{(n-1)+1} = 2^n.$$



Abbiamo visto con n bit quanti sono i simboli che possono essere rappresentati. Per rappresentare *N* simboli differenti quanti bit si devono usare?

Numero di bit necessari per rappresentare *N* oggetti
Dobbiamo fare in modo che la combinazione dei bit sia maggiore o uguale al numero dei simboli. In particolare ciò si traduce nella formula:

$$2^n \geq N$$

ossia dobbiamo scegliere il numero minimo di bit *n* per il quale 2^n è maggiore o al più uguale al numero di simboli che vogliamo rappresentare. Da questa formula, passando al logaritmo in base 2 entrambi i membri, otteniamo il valore di *n*:

$$n \geq \log_2 N$$

Evidentemente, $\log_2 N$ non sarà in generale un valore intero, quindi dobbiamo fare in modo che questa relazione ci produca un numero intero di bit.
Per cui scriveremo che *n* è la parte intera di $\log_2 N$ aggiungendo ancora 1 (arrotondiamo in eccesso all'intero più vicino). In alternativa basta cercare l'*n* per il quale 2^n è $\geq N$.

Es.: supponiamo di avere $N = 87$ oggetti e supponiamo di voler determinare il numero *n* di bit richiesto per rappresentare questi 87 oggetti:

$2^n \geq 87$
$n \geq \log_2 87$
$\log_2 87 = 6.65$ circa, per cui l'intero maggiore od uguale a 6.65 è $n = 7$ bit.
Per 87 oggetti occorrono 7 bit. Si noti che in realtà con $2^7 = 128$, ossia con 7 bit potremmo rappresentare 128 oggetti distinti e quindi abbiamo una certa discrezionalità nello scegliere le configurazioni, dobbiamo perciò stabilire con precisione quale configurazione è associata a ciascun oggetto. Alternativamente $2^6 = 64$ e $2^7 = 128$, ne consegue che ho bisogno di almeno $n = 7$ bit.

Supponiamo ad esempio di voler codificare tre colori: il Rosso, il Verde, il Blu (sono i tre colori fondamentali che permettono in un video di un computer di comporre tutti gli altri colori).
Abbiamo quindi $N = 3$; *n* si calcola tramite le formule precedenti e si ha $n = 2$.
Quindi abbiamo $2^2 = 4$ combinazioni, che sono: 00, 01, 10, 11
Abbiamo detto di voler codificare il R, il V, il B.

Possiamo allora scegliere per esempio la seguente corrispondenza:
00 R
01 V
10 B
11 combinazione non utilizzata

Oppure:
00 B
01 R
10 combinazione non utilizzata
11 V

Possiamo dunque scegliere più combinazioni.
Questo problema di univocità della rappresentazione è stato affrontato fin dai primordi dell'avvento del calcolatore. In particolare per i caratteri sono stati utilizzati i *codici di carattere*.

Codice ASCII
Uno dei più diffusi codice di caratteri è il *Codice ASCII* (ASCII è un acronimo e sta per *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange). È un codice di 7 bit che offre dunque la possibilità di esprimere 128 simboli diversi. In generale si utilizza il *codice ASCII esteso*, che utilizza 8 bit, e aggiunge altri 128 simboli specifici della nazione in cui ci si trova o del *sistema operativo* che si utilizza.

In alcuni casi i 7 bit sono utilizzati quando l'informazione viene trasferita da un calcolatore ad un altro dispositivo, ed allora minimizzare il numero di bit di rappresentazione vuol dire minimizzare il tempo di trasmissione dell'informazione stessa.
Il codice ASCII, è stato creato dallo *ANSI* (*A*merican *N*ational *S*tandardization *I*nstitute).

Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char
0	00000000	Null	32	00100000	Space	64	01000000	@	96	01100000	`
1	00000001	Start of heading	33	00100001	!	65	01000001	A	97	01100001	a
2	00000010	Start of text	34	00100010	"	66	01000010	B	98	01100010	b
3	00000011	End of text	35	00100011	#	67	01000011	C	99	01100011	c
4	00000100	End of transmit	36	00100100	$	68	01000100	D	100	01100100	d
5	00000101	Enquiry	37	00100101	%	69	01000101	E	101	01100101	e
6	00000110	Acknowledge	38	00100110	&	70	01000110	F	102	01100110	f
7	00000111	Audible bell	39	00100111	'	71	01000111	G	103	01100111	g
8	00001000	Backspace	40	00101000	(	72	01001000	H	104	01101000	h
9	00001001	Horizontal tab	41	00101001	)	73	01001001	I	105	01101001	i
10	00001010	Line feed	42	00101010	*	74	01001010	J	106	01101010	j
11	00001011	Vertical tab	43	00101011	+	75	01001011	K	107	01101011	k
12	00001100	Form feed	44	00101100	,	76	01001100	L	108	01101100	l
13	00001101	Carriage return	45	00101101	-	77	01001101	M	109	01101101	m
14	00001110	Shift out	46	00101110	.	78	01001110	N	110	01101110	n
15	00001111	Shift in	47	00101111	/	79	01001111	O	111	01101111	o
16	00010000	Data link escape	48	00110000	0	80	01010000	P	112	01110000	p
17	00010001	Device control 1	49	00110001	1	81	01010001	Q	113	01110001	q
18	00010010	Device control 2	50	00110010	2	82	01010010	R	114	01110010	r
19	00010011	Device control 3	51	00110011	3	83	01010011	S	115	01110011	s
20	00010100	Device control 4	52	00110100	4	84	01010100	T	116	01110100	t
21	00010101	Neg. acknowledge	53	00110101	5	85	01010101	U	117	01110101	u
22	00010110	Synchronous idle	54	00110110	6	86	01010110	V	118	01110110	v
23	00010111	End trans. block	55	00110111	7	87	01010111	W	119	01110111	w
24	00011000	Cancel	56	00111000	8	88	01011000	X	120	01111000	x
25	00011001	End of medium	57	00111001	9	89	01011001	Y	121	01111001	y
26	00011010	Substitution	58	00111010	:	90	01011010	Z	122	01111010	z
27	00011011	Escape	59	00111011	;	91	01011011	[	123	01111011	{
28	00011100	File separator	60	00111100	<	92	01011100	\	124	01111100	|
29	00011101	Group separator	61	00111101	=	93	01011101	]	125	01111101	}
30	00011110	Record separator	62	00111110	>	94	01011110	^	126	01111110	~
31	00011111	Unit separator	63	00111111	?	95	01011111	_	127	01111111	□

Tabella del codice ASCII (128 caratteri)

I primi 31 elementi del codice ASCII in realtà non sono caratteri ma rappresentano codici di controllo, ad esempio il codice associato al valore 8 rappresenta il *Backspace* che consente di cancellare l'ultimo carattere digitato, il codice associato al valore 10 indica fine linea e quello associato al valore 13 ritorno a capo. Il codice ASCII quindi utilizza i codici di controllo per la formattazione del testo, per la stampa del testo, o altre informazioni di questo tipo.

Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char
128	10000000	Ç	160	10100000	á	192	11000000	└	224	11100000	α
129	10000001	ü	161	10100001	í	193	11000001	┴	225	11100001	ß
130	10000010	é	162	10100010	ó	194	11000010	┬	226	11100010	Γ
131	10000011	â	163	10100011	ú	195	11000011	├	227	11100011	π
132	10000100	ä	164	10100100	ñ	196	11000100	─	228	11100100	Σ
133	10000101	à	165	10100101	Ñ	197	11000101	┼	229	11100101	σ
134	10000110	å	166	10100110	ª	198	11000110	╞	230	11100110	µ
135	10000111	ç	167	10100111	º	199	11000111	╟	231	11100111	τ
136	10001000	ê	168	10101000	¿	200	11001000	╚	232	11101000	Φ
137	10001001	ë	169	10101001	⌐	201	11001001	╔	233	11101001	Θ
138	10001010	è	170	10101010	¬	202	11001010	╩	234	11101010	Ω
139	10001011	ï	171	10101011	½	203	11001011	╦	235	11101011	δ
140	10001100	î	172	10101100	¼	204	11001100	╠	236	11101100	∞
141	10001101	ì	173	10101101	¡	205	11001101	═	237	11101101	ø
142	10001110	Ä	174	10101110	«	206	11001110	╬	238	11101110	ε
143	10001111	Å	175	10101111	»	207	11001111	╧	239	11101111	∩
144	10010000	É	176	10110000	░	208	11010000	╨	240	11110000	≡
145	10010001	æ	177	10110001	▒	209	11010001	╤	241	11110001	±
146	10010010	Æ	178	10110010	▓	210	11010010	╥	242	11110010	≥
147	10010011	ô	179	10110011	│	211	11010011	╙	243	11110011	≤
148	10010100	ö	180	10110100	┤	212	11010100	╘	244	11110100	⌠
149	10010101	ò	181	10110101	╡	213	11010101	╒	245	11110101	⌡
150	10010110	û	182	10110110	╢	214	11010110	╓	246	11110110	÷
151	10010111	ù	183	10110111	╖	215	11010111	╫	247	11110111	≈
152	10011000	ÿ	184	10111000	╕	216	11011000	╪	248	11111000	°
153	10011001	Ö	185	10111001	╣	217	11011001	┘	249	11111001	∙
154	10011010	Ü	186	10111010	║	218	11011010	┌	250	11111010	·
155	10011011	¢	187	10111011	╗	219	11011011	█	251	11111011	√
156	10011100	£	188	10111100	╝	220	11011100	▄	252	11111100	ⁿ
157	10011101	¥	189	10111101	╜	221	11011101	▌	253	11111101	²
158	10011110	₧	190	10111110	╛	222	11011110	▐	254	11111110	■
159	10011111	ƒ	191	10111111	┐	223	11011111	▀	255	11111111	

Tabella del codice ASCII esteso (successivi 128 caratteri)

Per comprendere meglio la codifica facciamo un esempio. La parola "CIAO" viene rappresentata all'interno di un elaboratore come sequenza di caratteri codificati in ASCII, come segue:

C	I	A	O
01000011	01001001	01000001	01001111
1° byte	2° byte	3° byte	4° byte

Le sequenze binarie hanno lo svantaggio d'essere molto lunghe in relazione alla quantità d'informazione che rappresentano. All'interno del computer non si può fare diversamente, ma per gli esseri umani ricordare e scrivere "0" ed "1" è un'attività tediosa oltre che difficile. Si cerca quindi di rappresentare in forme più concise tali sequenze, per rendere più snella l'operazione di scrittura. La rappresentazione esadecimale è la più utilizzata.

Per la codifica nella rappresentazione esadecimale si deve:
- suddividere la sequenza di bit in gruppi di 4 (da destra verso sinistra);
- ad ogni gruppo di 4 bit associare una cifra esadecimale secondo la corrispondenza definita di seguito:

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

Sulla base di queste considerazioni possiamo riscrivere la sequenza corrispondente a CIAO utilizzando la sequenza esadecimale: suddividiamo in 2 parti ciascun byte

C		I		A		O	
01000011		01001001		01000001		01001111	
0100	0011	0100	1001	0100	0001	0100	1111
4	3	4	9	4	1	4	F
43		49		41		4F	
1° byte		2° byte		3° byte		4° byte	

il primo byte è qui rappresentato mediante due cifre esadecimali. In modo analogo si ragiona per gli altri. La codifica finale è: 43 49 41 4F.

Rappresentazione dei numeri
Passiamo adesso alla codifica di un'altra classe di informazioni di grande rilevanza: i numeri. Nella codifica ASCII, chiamata anche "CODIFICA DI CARATTERE" (in quanto ad ogni carattere corrisponde un codice che utilizza un byte), possiamo rappresentare i numeri.
Per esempio 1237 è rappresentato come sequenza di 4 byte: un primo byte che contiene la codifica ASCII della cifra 1 (00110001), un secondo byte che contiene la codifica ASCII della cifra 2 (00110010), ..., un quarto byte che contiene la codifica ASCII della cifra 7 (00110111).
Questa forma di rappresentazione di un numero è idonea per la rappresentazione su video o per un testo ma *non è adatta per effettuare dei calcoli*. Possiamo, quindi, sinteticamente affermare che all'interno del nostro elaboratore abbiamo:

- codifica per i testi $\rightarrow$ ASCII
- codifica per i numeri

Dobbiamo innanzi tutto capire quale è la differenza fra la *rappresentazione di un numero* ed il suo *valore vero*. Per esempio, quando pensiamo al numero intero 4 come concetto astratto, possiamo rappresentarlo in modi diversi: con le dita di una mano, secondo la nostra codifica di sistema decimale, in numeri romani. Esistono quindi almeno tre rappresentazioni per la stessa entità.
In particolare delle tre rappresentazioni c'è ne è una che ha una certa rilevanza ed è quella usata comunemente da noi umani: il *sistema di rappresentazione decimale posizionale*.

Sistemi di rappresentazione posizionale e non
Il concetto della rappresentazione dei numeri mediante un sistema posizionale fu introdotto dagli arabi e permise di velocizzare notevolmente tutte le operazioni, soprattutto le moltiplicazioni.
Un *sistema posizionale* è *caratterizzato* da una *base* o *radice* che indicheremo con la lettera r e da un *insieme di cifre* d_i che vanno da 0 a (r-1).
Il sistema decimale ha una base $r = 10$ che utilizza 10 cifre: 0, 1, 2, …, 9.
Se avessimo $r = 3$ avremmo le seguenti cifre: 0, 1, 2.
Prima di vedere cosa succede all'interno di un elaboratore, vediamo come è possibile passare, in un sistema posizionale, dalla rappresentazione di un numero al suo valore vero (N_v) o viceversa.
In un sistema di rappresentazione posizionale il generico numero n è rappresentato da una sequenza di cifre:

$$N = d_m\ d_{m\text{-}1\ldots}\ d_1\ d_0$$

ove ciascuna di queste cifre appartiene all'insieme proprio del sistema di numerazione.

Per esempio se fosse $N = 1732$ il suo valore si otterrebbe tenendo presente che abbiamo 2 unità, 3 decine, 7 centinaia, 1 migliaio e dunque:

N = 1 migliaio + 7 centinaia + 3 decine + 2 unità

Ciò significa che all'entità 2 associamo un peso di 10^0, all'entità 3 il peso 10^1, all'entità 7 il peso 10^2, e all'entità 1 il peso 10^3.
Da questa semplice considerazione possiamo ricavare quella che è una *regola dei sistemi di rappresentazione posizionale* (posizionale perché ad ogni cifra è associato un ben determinato peso che dipende dalla posizione della cifra all'interno del numero). Da quanto detto è possibile dedurre alcune regole:

Sistemi posizionali
Assegnato un numero intero non negativo N, questo è rappresentato come sequenza di n simboli (cifre):

$$N = d_{n\text{-}1}\ d_{n\text{-}2}\ldots\ d_1\ d_0$$

Il valore vero è:

$$N_v = d_{n\text{-}1}\ r^{n\text{-}1} + d_{n\text{-}2}\ r^{n\text{-}2} + \ldots\ldots + d_1\ r^1 + d_0\ r^0$$

essendo:
r la base del sistema di numerazione,
d_i le cifre appartenenti all'insieme 0, 1, …, r-1.

Il *sistema di numerazione romano* non è posizionale. Infatti la sola pozione della cifra non dice nulla sul modo con cui dobbiamo trattarla ma dipende dal contesto. In pratica, consideriamo i due seguenti numeri utilizzando la notazione romana:

XII
XIX

corrispondenti rispettivamente ai numeri 12 e 19. Consideriamo la cifra "I" che occupa la seconda posizione di entrambe i numeri: nel primo caso la sommiamo al valore 10 (X) della cifra più a sinistra, nel secondo caso la sottraiamo al valore 10 della cifra seguente (IX rappresenta 9). Quindi la stessa cifra I in seconda posizione (da destra verso sinistra), è trattata in modo completamente diverso e assume regole diverse dipendentemente dal contesto: questo non è un sistema di riferimento posizionale.
È ben diverso scrivere 212 , 918, 19: il numero 1 in seconda posizione ha un valore intrinseco (ossia uno) ed è sempre associato alle decine (ossia per l'N_v ha un valore sempre pari a 10).

Conversione da una base qualunque alla base decimale
Assumiamo un sistema con base $r = 3$. Abbiamo a disposizione 3 cifre: 0,1,2.
Supponiamo di avere il numero $121_{(3)}$, ove il 3 a pedice indica la base (si ricordi che le cifre a disposizione sono 0, 1, 2). Tale numero ha valore vero:
$N_v = 1* 3^2 + 2* 3^1 + 1*3^0 = 1*9 +2*3 + 1*1 = 9 + 6 + 1 = 16_{(10)}$

Supponendo di avere il numero $1312_{(4)}$ il suo valore vero è:
$N_v = 1* 4^3 + 3* 4^2 + 1*4^1 + 2* 4^0 = 1*64 + 3*16 + 1*4 + 2*1 = 64 + 48 + 4 + 2 = 118_{(10)}$

Il *sistema binario* è un sistema avente come base $r = 2$ e cifre 0 e 1. Il sistema binario è compatibile con la tecnologia utilizzata nei calcolatore. Risulta dunque importante avere una certa dimestichezza con tale sistema. Per convertire un numero dal sistema binario al sistema decimale applichiamo la formula del valore vero. Ad esempio, dato il numero 1010 in base 2 si vuole trovare il suo valore in base 10:

$1010_{(2)} = 1* 2^3 + 0* 2^2 + 1*2^1 +0* 2^0 = 8 +2 = 10_{(10)}$

Il *sistema esadecimale* ha una base $r = 16$ avente come cifre i seguenti simboli: 0, 1, 2, …, 9, A, B, C, D, E, F con valori 0, 1, …, 9, 10, 11, 12, 13, 14, 15 rispettivamente.

Ad esempio: $10F2_{(16)} = 1 * 16^3 + 0 * 16^2 + F * 16^1 + 2* 16^0 = 1 * 4096 + 15 * 16^1 + 2*1 =$
$= 1*4096 + 15*16 + 2*1 = 4096 + 240 + 2 = 4338_{(10)}$

Una piccola nota sulla codifica di carattere ASCII
Ora che sappiamo ricavare il valore numerico assegnato ad una sequenza di bit è possibile comprendere appieno perché nella tabella ASCII è indicata la corrispondenza fra valore decimale, valore binario e carattere. Il valore decimale è l'N_v corrispondente alla sequenza binaria. Inoltre è importante notare che ogni lettera ha un valore ASCII maggiore delle lettere che la precedono nell'alfabeto. Ad esempio, poiché la B segue la lettera A la codifica ASCII assegna alla B un valore numerico ($01000010_{(2)} = 66_{(10)}$) maggiore di una unità rispetto a quello della lettera A ($01000001_{(2)} = 65_{(10)}$). In tal modo il problema di ordinare *n* lettere (o parole) è equivalente al problema di ordinare *n* numeri. Tale ordinamento è rispettato anche per la rappresentazione dei simboli 0, 1, …, 9.

Conversione dalla base 2 alla base 10
Dato un numero nel sistema decimale, vogliamo vedere come questo viene rappresentato all'interno dell'elaboratore in termini di 0 ed 1, ossia vogliamo effettuare la conversione da sistema decimale a sistema binario. Esistono due metodi che è possibile utilizzare.

Metodo delle potenze
Per utilizzare tale metodo si deve tenere presente la tabellina delle potenze di 2.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256

Supponendo di avere posto tutti i bit al valore zero, il metodo è il seguente:
1) si considera il numero $N_{(10)} > 0$ e si cerca fra le potenze di 2 quella più vicina a tale numero (minore o uguale), supponiamo che sia 2^n;
2) si pone ad uno il valore del bit di posto n (in realtà l'n+1-esimo bit)
3) si aggiorna il valore di $N_{(10)}$ sottraendogli 2^n ($N_{(10)} - 2^n \rightarrow N_{(10)}$)
4) se $N_{(10)}$ vale zero si termina, altrimenti si ritorna al punto 1).

Vediamo un esempio. Consideriamo il numero $N_{(10)} = 34_{(10)}$ e cerchiamo di trovare la sequenza di bit che lo rappresenta in base 2. A tal fine stabiliamo di avere a disposizione 8 bit.
1) $2^5 = 32$ ($< N_{(10)}$), $2^6 = 64$ ($> N_{(10)}$), quindi prendiamo $n = 5$.
2) accendiamo il bit di posto 5: 00**1**00000
3) $34 - 32 = 2$, ossia $N_{(10)} = 2$
4) $N_{(10)} > 0$, ritorniamo al punto 1) utilizzando il nuovo valore
1) $2^1 = 2$ ($= N_{(10)}$) e quindi scegliamo $n = 1$
2) accendiamo il bit di posto 1: 0010001**0**
3) $2 - 2 = 0$, ossia $N_{(10)} = 0$
4) $N_{(10)} = 0$ e si può interrompere la procedura.

Il numero binario è $00100010_{(2)}$, oppure, non considerando gli zeri del bit 6 e 7: $100010_{(2)}$

Per verificare la conversione possiamo ripassare dalla base 2 alla base 10.
Ad esempio nel primo caso si aveva:
$00100010_{(2)} = 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 32+2 = 34_{(10)}$.
Verificare, come esercizio, che $67_{(10)} = 01000011_{(2)}$.

Il procedimento visto può essere applicato a numeri relativamente piccoli ma in generale la metodologia più utilizzata è l'algoritmo per divisioni.

Algoritmo per divisioni
In realtà consente di passare dalla rappresentazione in base 10 ad una rappresentazione in una base r qualunque $N_{(10)} \rightarrow N_{(r)}$

Come sappiamo il valore di un numero è esprimibile mediante la relazione:
$N_{(10)} = d_{n-1} r^{n-1} + d_{n-2} r^{n-2} + \ldots + d_1 r^1 + d_0 r^0$

mettiamo a fattore comune r (la base) fra le prime n-1 entità:

$N_{(10)} = r\,(\,d_{n-1}\,r^{n-2} + d_{n-2}\,r^{n-3} + \ldots + d_1\,) + d_0$

ponendo la quantità entro le parentesi tonde $(\,d_{n-1}\,r^{n-2} + d_{n-2}\,r^{n-3} + \ldots + d_1\,) = Q_0$, possiamo scrivere:

$$N_{(10)} = r\,Q_0 + d_0$$

Tale espressione fa immediatamente pensare alla *definizione euclidea di divisione*, infatti:

$N_{(10)}$ è il *dividendo*
r è il *divisore*
Q_0 il *quoziente*
d_0 il *resto*.

Sappiamo che il dividendo è uguale al divisore per il quoziente più il resto, ossia:

$D = d\,Q + R$

Ne consegue che effettuando la divisione intera fra $N_{(10)}$ e il valore decimale della base r cui si vuole passare (ad esempio $2_{(10)}$) otterremo che il resto d_0 è la cifra meno significativa nella rappresentazione in base r. Si noti che tale cifra è sicuramente compresa nell'insieme 1, 2, …, $r - 1$ in quanto il resto di una divisione intera è sicuramente inferiore al divisore r.

Se consideriamo adesso:

$Q_0 = d_{n-1}\,r^{n-2} + d_{n-2}\,r^{n-3} + \ldots + d_1 = r\,(d_{n-1}\,r^{n-3} + d_{n-2}\,r^{n-4} + \ldots + d_2) + d_1 = r\,Q_1 + d_1$

possiamo ancora una volta paragonare $Q_0 = r\,Q_1 + d_1$ con la definizione euclidea, avendo individuato la cifra di posto 1 (d_1) nella nuova rappresentazione.
Iterando il procedimento ci fermeremo quando $Q_i = 0$.

Esempio: supponiamo di voler passare alla base $r = 2$ e che $N_{(10)} = 34_{(10)}$.

34:2 = 17 con $R = 0 \Rightarrow d_0 = 0$ (ossia utilizzando 8 bit: 0000000**0**$_{(2)}$)
17:2 = 8 con $R = 1 \Rightarrow d_1 = 1$ (ossia: 000000**1**0$_{(2)}$)
8:2 = 4 con $R = 0 \Rightarrow d_2 = 0$ (ossia: 00000**0**10$_{(2)}$)
4:2 = 2 con $R = 0 \Rightarrow d_3 = 0$ (ossia: 0000**0**010$_{(2)}$)
2:2 = 1 con $R = 0 \Rightarrow d_4 = 0$ (ossia: 000**0**0010$_{(2)}$)
1:2 = 0 con $R = 1 \Rightarrow d_5 = 1$ (ossia: 00**1**00010$_{(2)}$)
possiamo fermarci in quanto da ora in poi la divisione avrà quoziente sempre pari a zero e resto zero.
In definitiva il valore $34_{(10)} = 00100010_{(2)} = 100010_{(2)}$

Si provi come esercizio ad assumere $r = 2$ e $N_{(10)} = 79_{(10)}$.

Questo tipo di procedimento si può estendere a qualsiasi base.

Ad esempio: $r = 16$ e $N_{(10)} = 58506_{(10)}$.

58506:16=3656 $R = 10$ (che corrisponde alla cifra esadecimale A)

3656:16=228 $R = 8$
228:16=14 $R = 4$
14:16=0 $R = 14$ (che corrisponde alla cifra esadecimale E)

Quindi $58506_{(10)} = E48A_{(16)}$