

# Codifica delle Informazioni e Aritmetica Binaria

Reti Logiche T

Pierluigi Zama Ramirez  
Pierluigi.zama@unibo.it

# Programma di reti logiche

*Saper  
Descrivere  
Progettare  
e Analizzare  
Le  
Macchine  
Digitali*

5: Reti sequenziali sincrone





4: Reti sequenziali asincrone

3: Reti combinatorie

2: Codifica binaria dell'infor.

1: Macchine digitali

# Argomenti

- Informazione 
- Codifica Binaria *01*
- Codici 
- Rappresentare i numeri 
- Operazioni aritmetiche 

# Informazione

- Attributo del messaggio
- E' un entità misurabile
- E' diminuzione di incertezza (esprime una scelta tra un insieme di alternative possibili)
- Dato un segnale, l'informazione è contenuta nella sua variazione entro la sua dinamica.
- Nel corso di Reti Logiche i segnali sono digitali

# Informazione

- L'informazione può essere rappresentata in bit (Binary digIT): un'astrazione per il livello elettrico alto (1) o basso (0) [logica positiva].
- Quantità di informazione: il numero minimo di bit che servono a codificare un messaggio
- Un oggetto 'vero' o 'falso' porta 1 bit di informazione.
- Esempi di informazione: Testo, immagini, audio, video...
- Le *macchine digitali* usano l'informazione codificata in codice binario per dare dei risultati, anch'essi in codice binario

# Codice: Simboli, Alfabeto e Stringa

**Codice:** E' un sistema di **Simboli** elementari appartenenti ad un **Alfabeto** convenzionalmente designati per rappresentare un'informazione.

**Informazione:** **Stringa** di lunghezza finita di **Simboli**.

Esempi:

**Testo**

*Alfabeto: Caratteri*

*Simboli: {a,b,c....,z}*

*Esempio di stringa: «Ciao»*

***Numeri decimali***

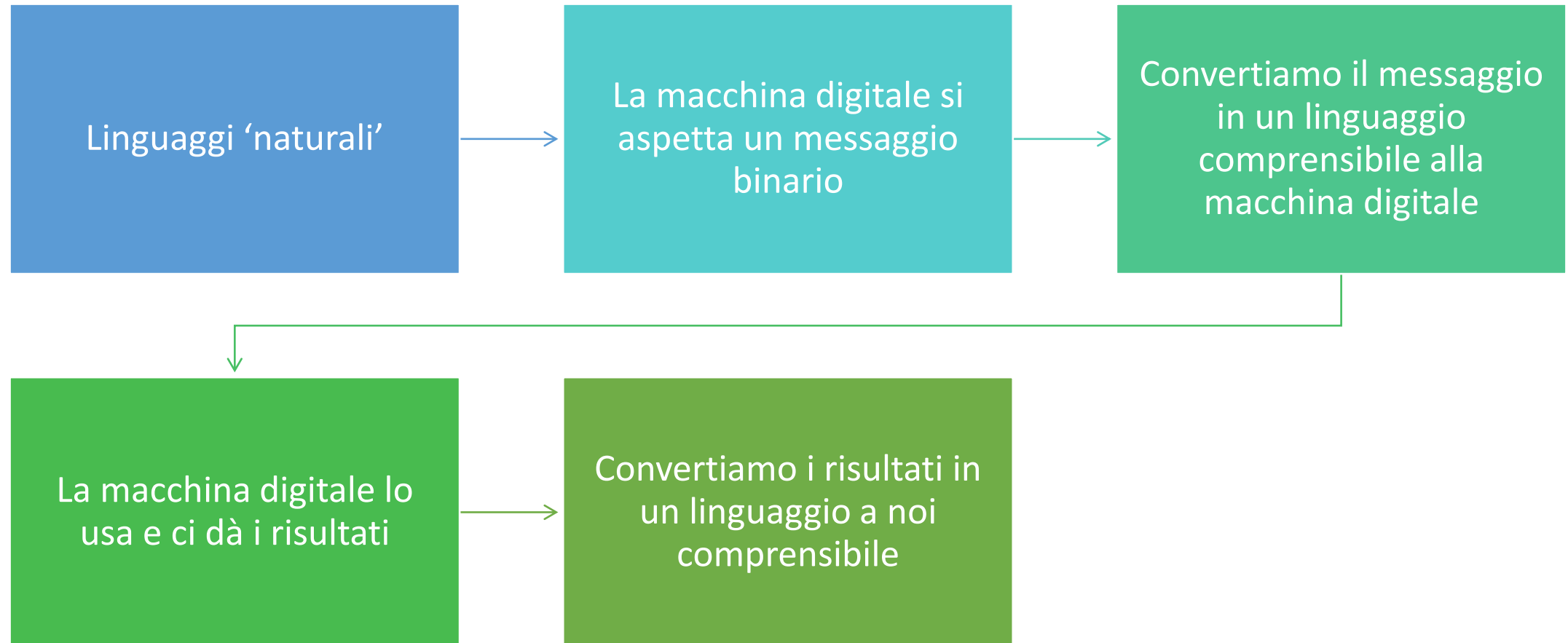
*Alfabeto: Cifre*

*Simboli: {0,1,2,3,4,5,6,7,8,9}*

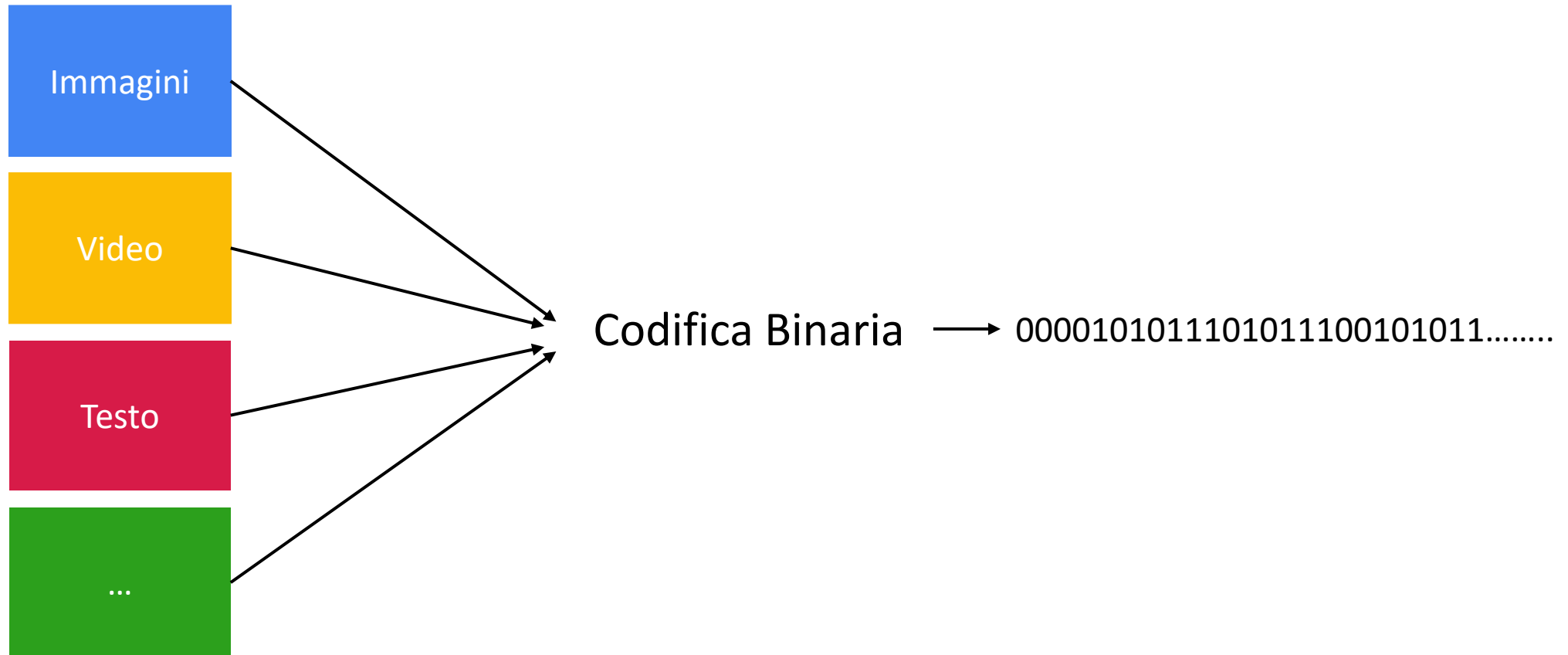
*Esempio di stringa: «519»*

**Nel codice binario l'alfabeto è formato dai simboli «0» e «1»**

# Codifica Binaria



# Codifica Binaria





# Codifica Binaria

I simboli vengono codificati tramite sequenze di 0 e 1

Per 2 simboli basta 1 bit

Per 4 simboli servono 2 bit

Per  $M$  simboli servono  $N$  bit:  $N = \lceil \log_2 M \rceil$

Alfabeto  $A_1 = \{0,1,2,3,4,5,6,7,8,9\}$ :  $\lceil \log_2 10 \rceil = 4$

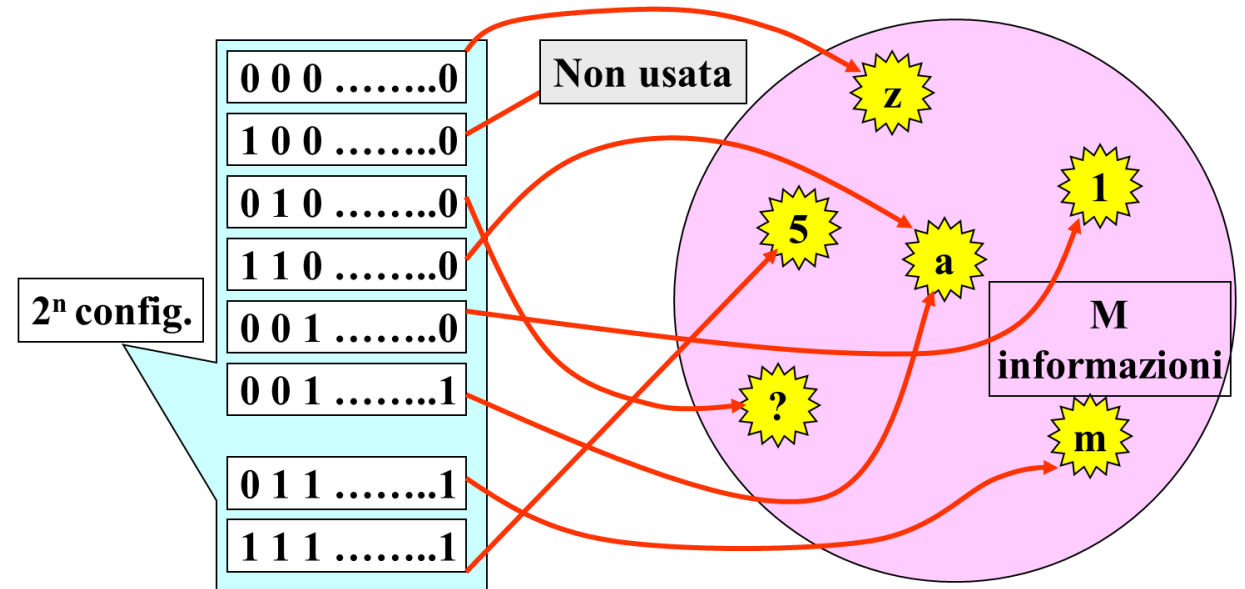
Occorrono *almeno* 4 bit.

*Potrei codificare fino a 16 simboli: da qui nasce la numerazione esadecimale*

Alfabeto latino (solo maiuscole)  $\lceil \log_2 26 \rceil = 5$

Occorrono *almeno* 5 bit.

Quanti bit per includere le minuscole e i numeri?



# Codice Binario

Una volta calcolato il numero di bit  $n$  necessario per codificare gli  $M$  simboli del mio alfabeto, devo far corrispondere ad ogni simbolo almeno una configurazione.

Il *codice binario* è una funzione che ad ogni configurazione di  $n$  bit fa corrispondere uno dei simboli dell'alfabeto.

$$2^n \geq M$$

*Perché  $\geq$ , e non solo  $=$  ?*

# Codice Binario

**La scelta del codice (la funzione che associa le configurazioni binarie ai simboli) deve essere condivisa.**

Dati  $M$  simboli e  $n$  bit, quanti possibili codici biunivoci (ogni simbolo 1 configurazione)?

$$C = \prod_{i=0}^{M-1} (2^n - i) = \frac{2^n!}{(2^n - M)!}$$

Con  $n$  numero di bit,  $M$  numero di simboli.

Per esempio, per  $n = 2$  bit, e  $M = 2$  simboli

$$C = 12$$

# Codice Binario

- ***Ridondante***

$$n > \lceil \log_2 M \rceil$$

Permettono il controllo e una maggiore resistenza all'errore; semplificano la generazione e l'interpretazione delle informazioni.

*(Potrei usare  $n$  bit, ma ne uso di più)*

- ***Non ridondante***

$$n = \lceil \log_2 M \rceil$$

Minor costo di memorizzazione e maggiore efficienza di calcolo (e alcune situazioni particolari)

- ***Proprietario***

Volto all'ottimizzazione delle prestazioni di una specifica macchina e/o a 'blindare' il mercato.  
Esempi: Linguaggio Assembler, Telecomando TV

- ***Standard***

Esito dell'attività di un ente internazionale, o codice proprietario largamente riconosciuto e utilizzato.

Esempi: Stampanti e Calcolatori

# Rappresentare i numeri: Codice Binary Code Decimal (BCD)

Cifra Decimale	Codice Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

**Codice che rappresenta ogni cifra decimale con 4 bit**

$$\lceil \log_2 10 \rceil = 4 \text{ bit}$$

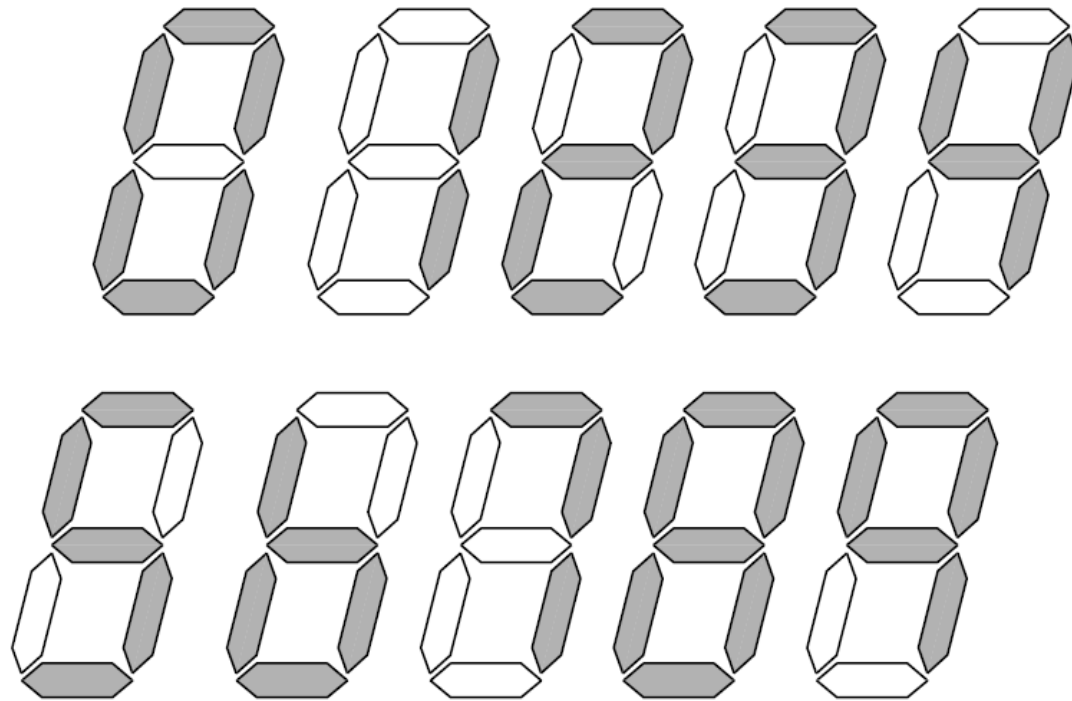
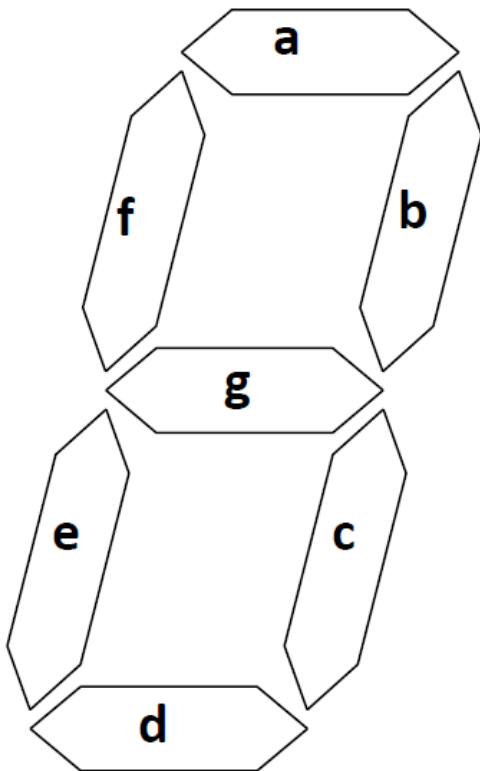
Numeri con k cifre: 4k bit

Codifichiamo il numero 371 (3 cifre = 12 bit):

0011 0111 0001

# Rappresentare i numeri: Codice a 7 segmenti

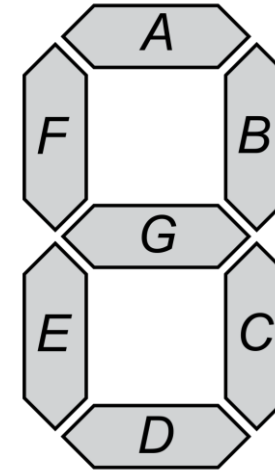
**Codice ridondante utilizzato per visualizzare a display numeri decimali**



	a	b	c	d	e	f	g
<b>Zero</b>	1	1	1	1	1	1	0
<b>Uno</b>	0	1	1	0	0	0	0
<b>Ecc.</b>							

# Rappresentare i numeri: Codice a 7 segmenti

Cifra Decimale	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0(1)	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1



**Codice ridondante utilizzato per visualizzare a display numeri decimali**

$M = 10$  (cifre decimali)  $n = 7 > 4$

# Rappresentare i numeri: Codice 1 of n

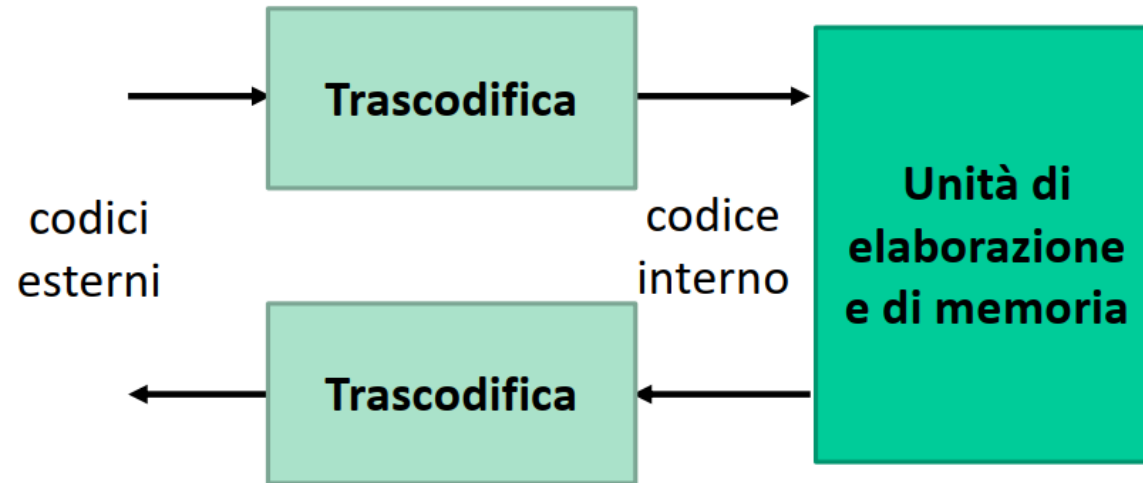
Cifra Decimale	#1	#2	#3	#4	#5
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

**Codifichiamo n simboli con n bit.**

L'n-esimo simbolo viene codificato con n-1 bit a zero,  
e un bit a uno, alla posizione n



# Conversione tra codici - Trascodifica



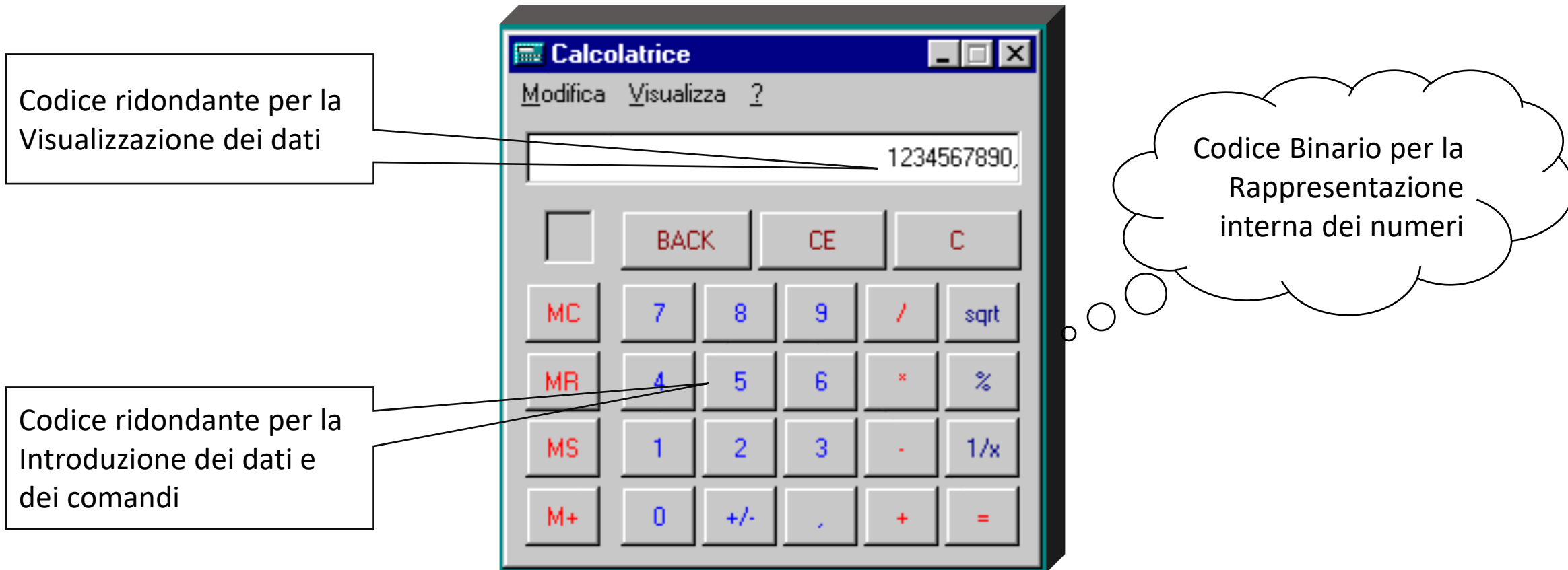
La conversione tra codici viene detta transcodifica.

Codice esterno:

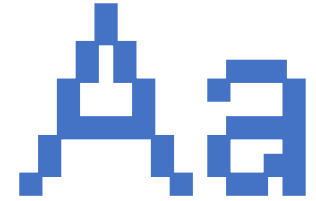
- **ridondante**, per semplificare la generazione e l'interpretazione delle informazioni,
- **standard**, per rendere possibile la connessione di macchine (o unità di I/O) realizzate da costruttori diversi.

Codice interno: è di norma **non ridondante** per minimizzare il numero di bit da elaborare e da memorizzare.

# Esempio di Trascodifica: Calcolatrice Tascabile



# ASCII e Unicode



American Standard Code for Information Interchange

ASCII control characters			ASCII printable characters									Extended ASCII characters								
DEC	HEX	Simbolo ASCII	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
00	00h	NULL (carácter nulo)	32	20h	espacio	64	40h	@	96	60h	`	128	80h	Ç	160	A0h	á	192	C0h	Ł
01	01h	SOH (inicio encabezado)	33	21h	!	65	41h	A	97	61h	a	129	81h	ü	161	A1h	î	193	C1h	ł
02	02h	STX (inicio texto)	34	22h	"	66	42h	B	98	62h	b	130	82h	é	162	A2h	ó	194	C2h	ł
03	03h	ETX (fin de texto)	35	23h	#	67	43h	C	99	63h	c	131	83h	â	163	A3h	ô	195	C3h	ł
04	04h	EOT (fin transmisión)	36	24h	\$	68	44h	D	100	64h	d	132	84h	ä	164	A4h	ñ	196	C4h	ł
05	05h	ENQ (enquiry)	37	25h	%	69	45h	E	101	65h	e	133	85h	à	165	A5h	Ñ	197	C5h	ł
06	06h	ACK (acknowledgement)	38	26h	&	70	46h	F	102	66h	f	134	86h	á	166	A6h	ª	198	C6h	ł
07	07h	BEL (timbre)	39	27h	'	71	47h	G	103	67h	g	135	87h	ç	167	A7h	º	199	C7h	ł
08	08h	BS (retroceso)	40	28h	(	72	48h	H	104	68h	h	136	88h	ê	168	A8h	¿	200	C8h	ł
09	09h	HT (tab horizontal)	41	29h	)	73	49h	I	105	69h	i	137	89h	ë	169	A9h	¸	201	C9h	ł
10	0Ah	LF (salto de línea)	42	2Ah	*	74	4Ah	J	106	6Ah	j	138	8Ah	è	170	AAh	¬	202	CAh	ł
11	0Bh	VT (tab vertical)	43	2Bh	+	75	4Bh	K	107	6Bh	k	139	8Bh	ï	171	ABh	½	203	CBh	ł
12	0Ch	FF (form feed)	44	2Ch	,	76	4Ch	L	108	6Ch	l	140	8Ch	î	172	ACH	¼	204	CCh	ł
13	0Dh	CR (retorno de carro)	45	2Dh	-	77	4Dh	M	109	6Dh	m	141	8Dh	ï	173	ADh	»	205	CDh	ł
14	0Eh	SO (shift Out)	46	2Eh	.	78	4Eh	N	110	6Eh	n	142	8Eh	Ä	174	AEh	«	206	CEh	ł
15	0Fh	SI (shift In)	47	2Fh	/	79	4Fh	O	111	6Fh	o	143	8Fh	Å	175	AFh	»	207	CFh	ł
16	10h	DLE (data link escape)	48	30h	0	80	50h	P	112	70h	p	144	90h	É	176	B0h	»	208	D0h	ł
17	11h	DC1 (device control 1)	49	31h	1	81	51h	Q	113	71h	q	145	91h	æ	177	B1h	»	209	D1h	ł
18	12h	DC2 (device control 2)	50	32h	2	82	52h	R	114	72h	r	146	92h	Æ	178	B2h	»	210	D2h	ł
19	13h	DC3 (device control 3)	51	33h	3	83	53h	S	115	73h	s	147	93h	ø	179	B3h	»	211	D3h	ł
20	14h	DC4 (device control 4)	52	34h	4	84	54h	T	116	74h	t	148	94h	ö	180	B4h	»	212	D4h	ł
21	15h	NAK (negative acknowle.)	53	35h	5	85	55h	U	117	75h	u	149	95h	õ	181	B5h	»	213	D5h	ł
22	16h	SYN (synchronous idle)	54	36h	6	86	56h	V	118	76h	v	150	96h	ù	182	B6h	»	214	D6h	ł
23	17h	ETB (end of trans. block)	55	37h	7	87	57h	W	119	77h	w	151	97h	û	183	B7h	»	215	D7h	ł
24	18h	CAN (cancel)	56	38h	8	88	58h	X	120	78h	x	152	98h	ÿ	184	B8h	»	216	D8h	ł
25	19h	EM (end of medium)	57	39h	9	89	59h	Y	121	79h	y	153	99h	Ö	185	B9h	»	217	D9h	ł
26	1Ah	SUB (substitute)	58	3Ah	:	90	5Ah	Z	122	7Ah	z	154	9Ah	Ü	186	BAh	»	218	DAh	ł
27	1Bh	ESC (escape)	59	3Bh	;	91	5Bh	[	123	7Bh	{	155	9Bh	ø	187	BBh	»	219	DBh	ł
28	1Ch	FS (file separator)	60	3Ch	<	92	5Ch	\	124	7Ch	}	156	9Ch	£	188	BBh	»	220	DBh	ł
29	1Dh	GS (group separator)	61	3Dh	=	93	5Dh	]	125	7Dh	}	157	9Dh	Ø	189	BDh	»	221	DDh	ł
30	1Eh	RS (record separator)	62	3Eh	>	94	5Eh	^	126	7Eh	~	158	9Eh	x	190	BEh	»	222	DEh	ł
31	1Fh	US (unit separator)	63	3Fh	?	95	5Fh	-				159	9Fh	f	191	BFh	»	223	DFh	ł
127	20h	DEL (delete)																		

[theasciicode.com.ar](http://theasciicode.com.ar)

Con 8 bit, il codice (Extended) ASCII codifica tutte le possibili lingue originate dall'alfabeto latino, i numeri, la punteggiatura.  
Con 16 bit, il codice Unicode codifica (quasi) tutte le lingue conosciute sulla Terra, mantenendo la compatibilità con il codice ASCII.

Il tipo di dato 'char' del C è adibito ai caratteri ASCII.

# Bitmap: un codice ridondante per simboli alfanumerici

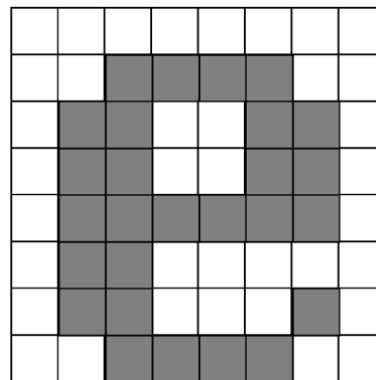
Rappresentazione binaria dei simboli **grafici** dei caratteri

Mentre nelle stampanti «ad impatto» basta il codice ASCII, per molti altri casi (getto d'inchiostro, laser, ma anche per mostrare a monitor) è necessario definire dei simboli grafici mediante matrici di pixel (***bitmap***)

Ciascun elemento di una bitmap è rappresentato da un insieme di bit

- Bianco/nero: 1 pixel -> 1 bit
- Scala di grigi: 1 pixel -> 8 bit
- Colori RGB: 1 pixel -> 3x8 bit

**Font:** insieme dei simboli grafici caratterizzati da un certo stile



Esempio bitmap 8x8

# Distanza

## Distanza tra due configurazioni binarie di n bit

- È il numero di bit omologhi con valore diverso:  
 $D(000,111) = 3$                        $D(011,010)=1$

## Distanza minima

- Il numero minimo DMIN che assume la distanza entro le configurazioni di un codice.

## Nota bene

- I codici non ridondanti hanno sempre  $DMIN=1$
- I codici ridondanti (e.g., 1-of-n) possono avere  $DMIN \geq 1$

# Codice Gray

E' un codice binario, composto da un qualsiasi numero di bit maggiore di due, nel quale il passaggio da un numero rappresentato al successivo (o precedente) comporta la variazione di un solo bit

Non lo usiamo per 'contare', ma per denominare degli 'stati' in un sistema.

Passando da 01 a 10 forse è meglio non rischiare che si passi per 11!

Stato	Codifica Gray
Bomba spenta	00
Bomba innescata	01
Boom!	11
Bomba disinnescata	10

# Numeri in base B

$$n_B = \sum_{i=-\infty}^{+\infty} a_i B^i \quad a_i \in \text{Alfabeto}$$

Esempi base B=10:

$$3724,28_{10} = 0 \cdot 10^4 + 3 \cdot 10^3 + 7 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 + 2 \cdot 10^{-1} \\ + 8 \cdot 10^{-2} + 0 \cdot 10^{-3}$$

Esattamente allo stesso modo

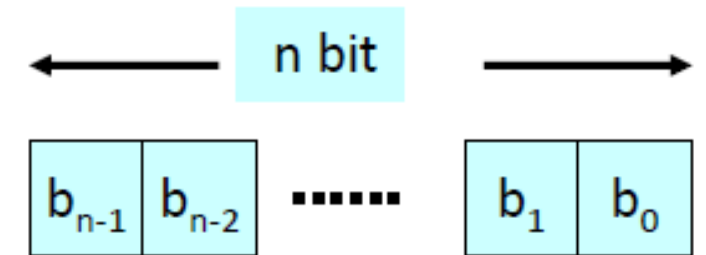
$$49_{10} = 110001_2$$

# Numeri in base 2

$$N = \sum_{i=0}^{n-1} b_i 2^i \quad b_i \in \{0,1\}$$

$i = 0$  perché **qui consideriamo i numeri interi**

$n$  è il numero di bit che vogliamo usare:



$$4_{10} = 100_2 = \sum_{i=0}^{3-1} a_i 2^i = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$$



# Quanti bit servono minimo?

Dato un numero  $Q$ , di quanti bit avrò bisogno per rappresentarlo?

$$N = \lceil \log_2 Q \rceil$$

La formula è sempre uguale.

Nei computer a 64 bit, una variabile numerica intero senza segno (unsigned long int) a 64 bit potrà contenere i numeri da 0 a  $2^{64}-1$

# Conversione Da base 2 a base 10

$$\begin{aligned} 100111101_2 &= \underbrace{1 \cdot 2^0}_{\text{bit 0}} + \underbrace{0 \cdot 2^1}_{\text{bit 1}} + \underbrace{1 \cdot 2^2}_{\text{bit 2}} + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + \\ &\quad + 0 \cdot 2^6 + 0 \cdot 2^7 + 1 \cdot 2^8 = \\ &= 1 + 4 + 8 + 16 + 32 + 256 = 317_{10} \end{aligned}$$

# Conversione Da base 10 a base 2

$$317 / 2 = 158 + \text{resto } 1$$

$$158 / 2 = 79 + \text{resto } 0$$

$$79 / 2 = 39 + \text{resto } 1$$

$$39 / 2 = 19 + \text{resto } 1$$

$$19 / 2 = 9 + \text{resto } 1$$

$$9 / 2 = 4 + \text{resto } 1$$

$$4 / 2 = 2 + \text{resto } 0$$

$$2 / 2 = 1 + \text{resto } 0$$

$$1 / 2 = 0 + \text{resto } 1$$

Ripercorrendo dal basso i resti delle divisioni:

100111101

**Il metodo funziona in generale (anche per altre coppie di basi) e solo per la parte intera, per la parte decimale c'è un algoritmo separato!!**

# Rappresentazione in Codice Esadecimale

**Sistema esadecimale:**  $B = 16$

**Cifre:** 0,1,...,9,a,b,c,d,e,f

n° di bit per cifra: 4 perché  $16 = 2^4$

Dec	Hex	Bin	Dec	Hex	Bin
0	0H	0000	8	8H	1000
1	1H	0001	9	9H	1001
2	2H	0010	10	AH	1010
3	3H	0011	11	BH	1011
4	4H	0100	12	CH	1100
5	5H	0101	13	DH	1101
6	6H	0110	14	EH	1110
7	7H	0111	15	FH	1111

Esempio:

$$76_{10} = 4CH = 0x4C$$

## **Conversione da decimale a esadecimale**

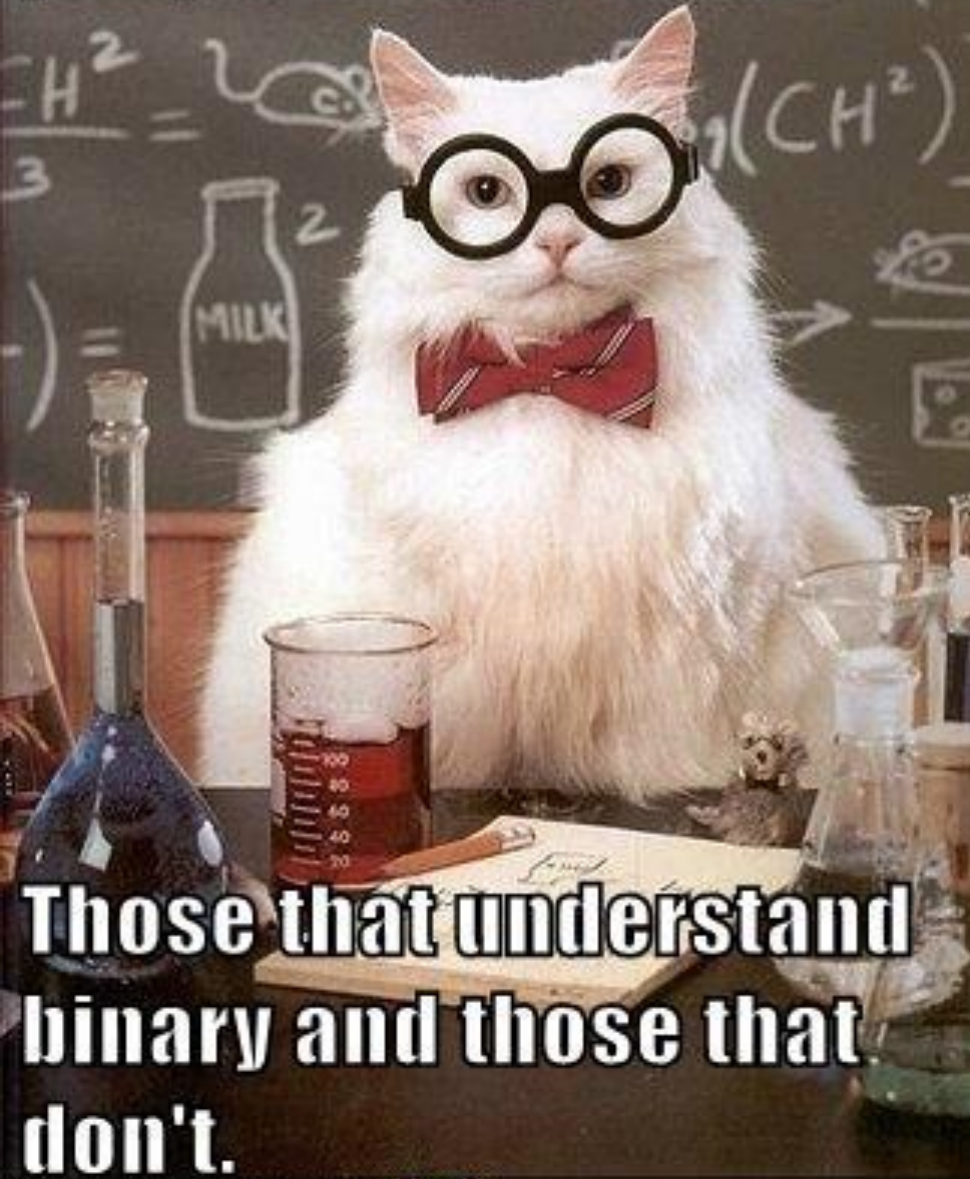
Stessa metodologia della slide precedente:

- $76/16 = 4$ , remainder is 12 (C)
- $4/16 = 0$ , remainder is 4

## **Conversione da binario a esadecimale**

1. Numero in binario  $76_{10} = 1001100_2$
2. Raggruppo 4 bit per 4: 0100 1100
3. Riscrivo la combinazione: 4C

**There are 10 types of  
people in this world.**



**Those that understand  
binary and those that  
don't.**

# Operazioni aritmetiche

# Somma aritmetica

Come è noto:

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

$$1+1=0$$

Senza dimenticare i riporti...

Gli oggetti che fanno questa operazione si chiamano *Half Adder* e *Full Adder*.

# (E)XOR: OR esclusivo

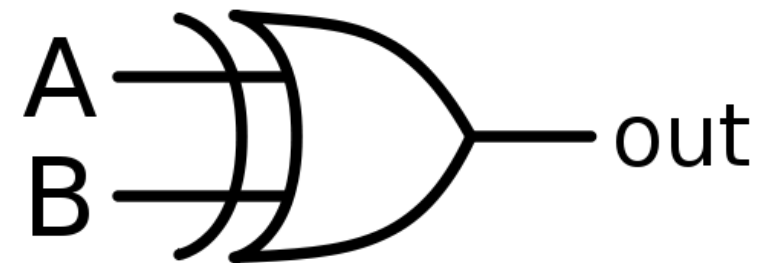
**Tabella della verità EXOR**

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Non fa parte dell'algebra della commutazione, che è composta da

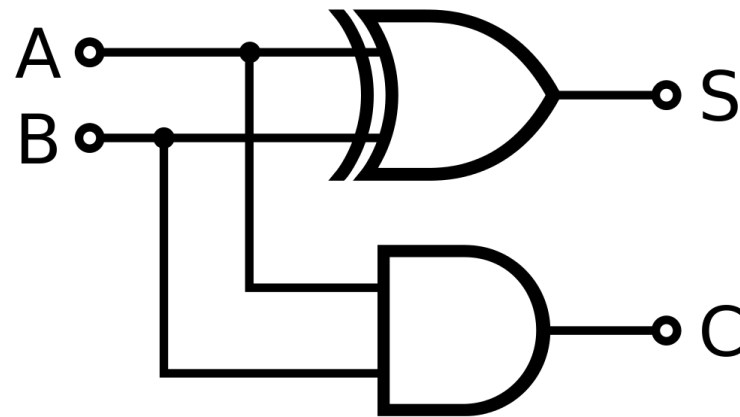
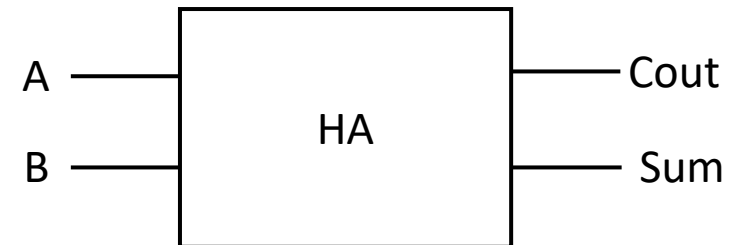
AND    OR    NOT

$$A \oplus B = (A + B)(\bar{A} + \bar{B})$$



# Half Adder

A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





# Full Adder (FA)

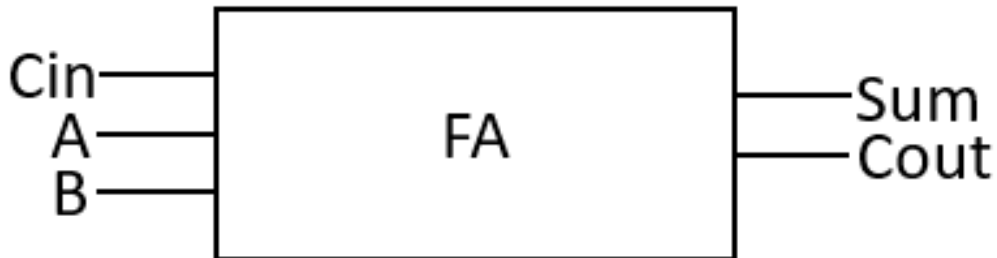
Somma fra due bit: non basta per eseguire la somma in colonna

$$0+0 = 00$$

$$0+1 = 01$$

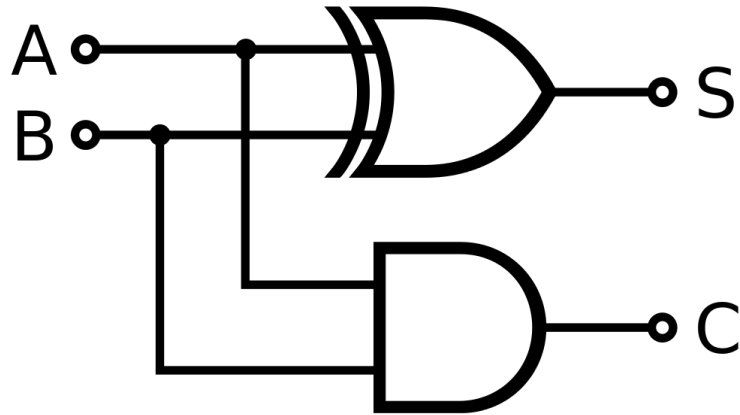
$$1+0 = 01$$

$$1+1 = \textcolor{red}{1}0$$

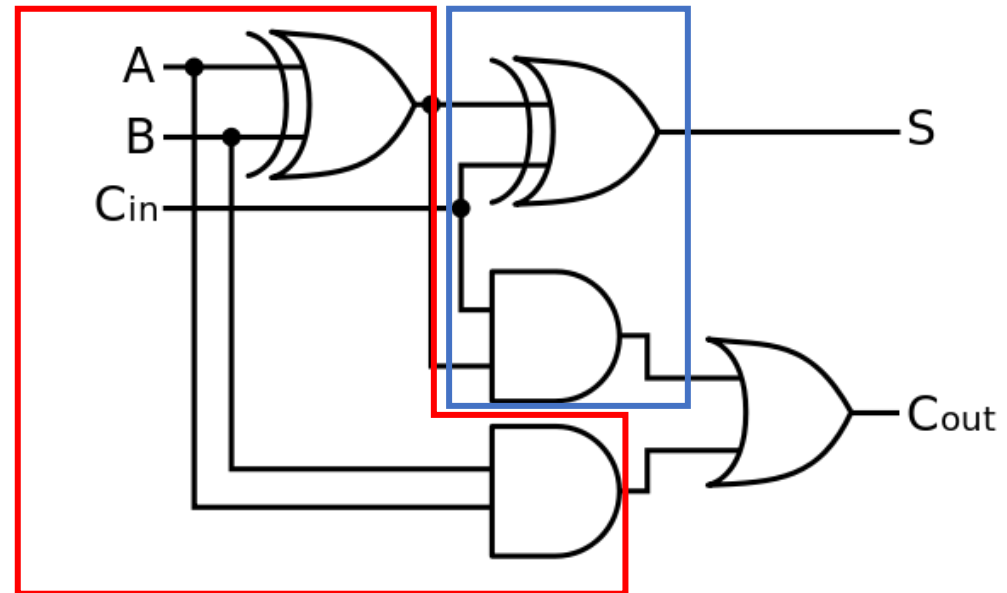


Cin	A	B	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# Full Adder (FA)



**Half Adder**

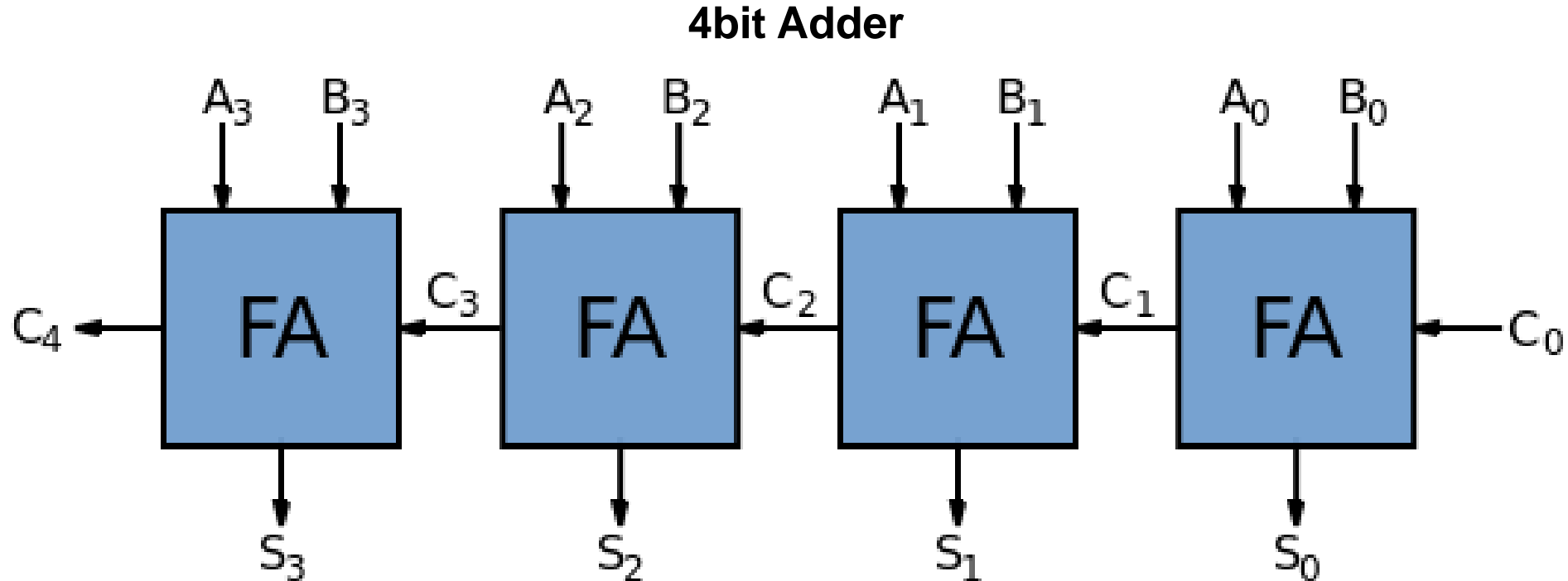


**Full Adder**

*Due Half Adder in cascata, più un OR*

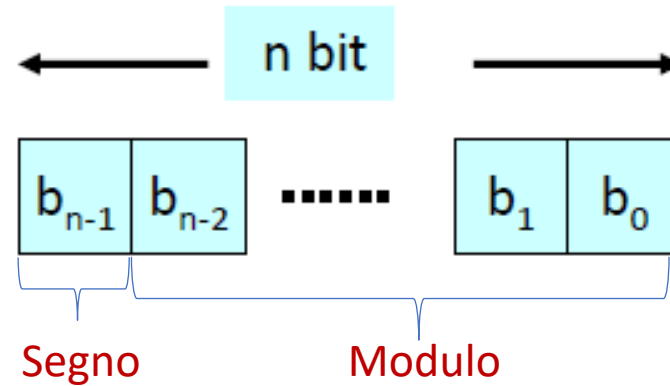
Il Full Adder consente di sommare i bit in una determinata posizione di una somma con due addendi, n Full Adder collegati consentono di ottenere un sommatore da n bit

# Adder per numeri senza segno a n bit



C<sub>4</sub> qui viene detto “overflow”: indica se la somma di peso maggiore ha generato riporto, e quindi se siamo usciti dal range di n bit.

# Numeri relativi: Rappresentazione segno e valore assoluto



Utilizziamo un bit per indicare il segno del dato.

Anche se è facile da leggere, si vede che qualcosa non torna:  $B + (-B) \neq 0$

$A + B$  non si ottiene la somma algebrica se  $B < 0$

Due rappresentazioni di zero

Range:  $[-2^{n-1} + 1, 2^{n-1} - 1]$

Segno		Decimale
0	00	0
0	01	+1
0	10	+2
0	11	+3
1	00	0
1	01	-1
1	10	-2
1	11	-3

# Complemento a 2 di un numero naturale $N$ formato da $n$ bit

$${}^2N \stackrel{\text{def}}{=} 2^n - N$$

**Complemento e incremento di  $N$  si ottiene  ${}^2N$**

- ${}^2N \stackrel{\text{def}}{=} 2^n - N$
- $= \underbrace{2^n - 1 - N}_{N'} + 1$   $N'$  = not logico di  $N$  in base 2
- ${}^2N = N' + 1$

Esempio con  $n = 3$  bit:

$$5_{10} = 101_2$$

$${}^25 \stackrel{\text{def}}{=} 2^3 - 5 = 2^3 - 1 - 5 + 1$$

In binario:  $(1)000 - 001 - 101 + 001 = 111 - 101 + 001 = \mathbf{010} + 001 = 011$

**Calcolo di  $A - B$  con  $A \geq B$**

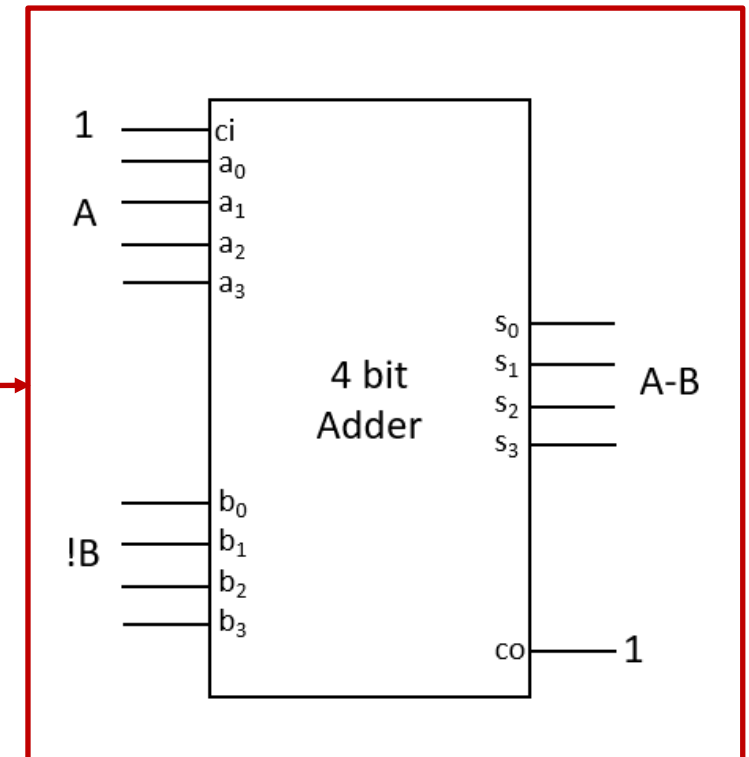
- $A - B = A - \underbrace{B + 2^n}_{2^B} - 2^n$
- $A - B = A + {}^2B - 2^n$
- $= A + B' + 1 - 2^n$

Notare che per fare una differenza continuiamo ad usare un Adder!

Il  $-2^n$  non viene considerato (sarebbe il segnale co)

**Inoltre se  $A=0$**

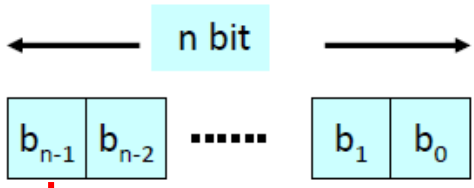
- $-B = {}^2B - 2^n = B' + 1 - 2^n$



# Numeri relativi: Rappresentazione in complemento a 2 con n bit

$N \geq 0$  : segno-valore assoluto

$N < 0$ :  $2^i(-N)$  con  $-N$  espresso in segno-valore assoluto



$$N = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Se  $b_i = 1 \forall i \rightarrow \sum_{i=0}^{n-2} 2^i = 2^{n-1} - 1$

Range:  $N \in [-2^{n-1}, 2^{n-1} - 1]$

B3	B2	B1	B0	N
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Il bit più significativo indica il **segno** (0:positivo, 1:negativo) come nella rappresentazione segno-valore assoluto. Tuttavia la lettura del numero è meno intuitiva.

# Alcune proprietà della rappresentazione in complemento a 2

*Dati A e B due numeri rappresentati in complemento a 2*

**1)  $-A = A' + 1$ . Si ottiene il negativo di un numero tramite inversione e incremento.**

$$\begin{array}{rcl} A: & 0001 & (+1) \\ & 1110 & + \\ & 1 & \\ -A: & 1111 & (-1) \end{array}$$

$$\begin{array}{rcl} A: & 1001 & (-7) \\ & 0110 & + \\ & 1 & \\ -A: & 0111 & (+7) \end{array}$$

**2)  $A + (-A) = 0$ . La somma di A e il suo negativo fa 0.**

$$\begin{array}{rcl} A (+1): & 0001 & + \\ -A (-1): & 1111 & = \\ & 0000 & \end{array}$$

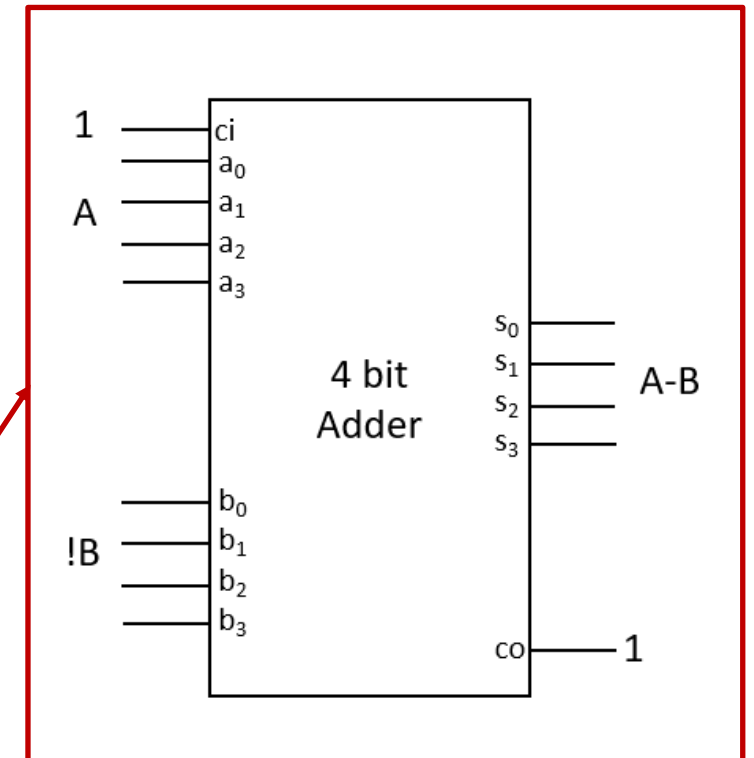
# Alcune proprietà della rappresentazione in complemento a 2

**3)  $A + B$  restituisce la somma algebrica rappresentata in complemento a 2:**

$A = -3$	1 1 0 1 +
$B = -4$	1 1 0 0 =
$A + B = -7$	1 0 0 1

$A = -4$	1 1 0 1 +
$B = +5$	0 1 0 1 =
$A + B = +1$	0 0 0 1

**4)  $A - B = A + B' + 1$ . Per sommare o sottrarre due numeri relativi espressi in complemento a 2 è sufficiente un addizionatore.**





# Confronto fra le rappresentazioni in relazione alla somma binaria

Config	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Unsign.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M&S	0	1	2	3	4	5	6	7	0	-1	-2	-3	-4	-5	-6	-7
Compl2	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1

Il sommatore comprende(capisce) solo la prima riga, esso parte dalla configurazione al suo primo ingresso e si muove in avanti di tante colonne quanto è il valore unsigned(seconda riga) corrispondente al suo secondo ingresso. Se supera 1111 allora continua da 0000 con carry\_out CO=1

In modulo e segno le configurazioni dei bit sono semplicemente leggibili per l'uomo, non si può usare un sommatore convenzionale con notazione in modulo e segno

La notazione usata nei calcolatori è quella in complemento a 2 che consente al sommatore tradizionale di eseguire la somma algebrica fra due quantità. L'esempio in:

- blu 5 -4. Ris= 1, Ov= 0, CO =1, Co<sub>n-2</sub>=1
- verde -2 + 4. Ris= 2, Ov=0, CO = 1, Co<sub>n-2</sub>=1
- rosso -4 -5. Ris= 7, Ov= 1, CO=1, Co<sub>n-2</sub>=0
- arancione 6+3. Ris= -7, Ov= 1, CO=0, Co<sub>n-2</sub>=1

Si può avere overflow quando si sommano quantità dello stesso segno, se c'è overflow il risultato avrà segno opposto. Studiando bene a partire da questa affermazione e dalla struttura interna dell'adder si arriva alla formula  $Ov = CO \text{ XOR } Co_{n-2}$

# Confronto fra le rappresentazioni in relazione alla somma binaria

Config	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Unsign.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M&S	0	1	2	3	4	5	6	7	0	-1	-2	-3	-4	-5	-6	-7
Compl2	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1

Il sommatore comprende(capisce) solo la prima riga, esso parte dalla configurazione al suo primo ingresso e si muove in avanti di tante colonne quanto è il valore unsigned(seconda riga) corrispondente al suo secondo ingresso. Se supera 1111 allora continua da 0000 con carry\_out CO=1

In modulo e segno le configurazioni dei bit sono semplicemente leggibili per l'uomo, non si può usare un sommatore convenzionale con notazione in modulo e segno

La notazione usata nei calcolatori è quella in complemento a 2 che consente al sommatore tradizionale di eseguire la somma algebrica fra due quantità. L'esempio in:

- blu 5 -4. Ris= **1**, Ov= 0, CO =1, Co<sub>n-2</sub>=1
- verde -2 + 4. Ris= **2**, Ov=0, CO = 1, Co<sub>n-2</sub>=1
- rosso -4 -5. Ris= **7**, Ov= 1, CO=1, Co<sub>n-2</sub>=0
- arancione 6+3. Ris= **-7**, Ov= 1, CO=0, Co<sub>n-2</sub>=1

Si può avere **overflow** quando si sommano quantità dello stesso segno, se c'è overflow il risultato avrà segno opposto. Studiando bene a partire da questa affermazione e dalla struttura interna dell'adder si arriva alla formula  $Ov = CO \text{ XOR } Co_{n-2}$

# Confronto fra le rappresentazioni in relazione alla somma binaria

Config	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Unsign.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M&S	0	1	2	3	4	5	6	7	0	-1	-2	-3	-4	-5	-6	-7
Compl2	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1

Il sommatore comprende(capisce) solo la prima riga, esso parte dalla configurazione al suo primo ingresso e si muove in avanti di tante colonne quanto è il valore unsigned(seconda riga) corrispondente al suo secondo ingresso. Se supera 1111 allora continua da 0000 con carry\_out CO=1

In modulo e segno le configurazioni dei bit sono semplicemente leggibili per l'uomo, non si può usare un sommatore convenzionale con notazione in modulo e segno

La notazione usata nei calcolatori è quella in complemento a 2 che consente al sommatore tradizionale di eseguire la somma algebrica fra due quantità. L'esempio in:

- blu 5 -4. Ris= **1**, Ov= 0, CO =1, Co<sub>n-2</sub>=1
- verde -2 + 4. Ris= **2**, Ov=0, CO = 1, Co<sub>n-2</sub>=1
- rosso -4 -5. Ris= **7**, Ov= 1, CO=1, Co<sub>n-2</sub>=0
- arancione 6+3. Ris= **-7**, Ov= 1, CO=0, Co<sub>n-2</sub>=1

**OVERFLOW: Errore ottenuto quando si ottiene un risultato al di fuori del range rappresentato.**

**OVERFLOW in complemento a 2!**

Si può avere overflow quando si sommano quantità dello stesso segno, se c'è overflow il risultato avrà segno opposto. Studiando bene a partire da questa affermazione e dalla struttura interna dell'adder si arriva alla formula  $Ov = CO \text{ XOR } Co_{n-2}$

# Confronto fra le rappresentazioni in relazione alla somma binaria

Config	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Unsign.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M&S	0	1	2	3	4	5	6	7	0	-1	-2	-3	-4	-5	-6	-7
Compl2	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1

Il sommatore comprende(capisce) solo la prima riga, esso parte dalla configurazione al suo primo ingresso e si muove in avanti di tante colonne quanto è il valore unsigned(seconda riga) corrispondente al suo secondo ingresso. Se supera 1111 allora continua da 0000 con carry\_out CO=1

In modulo e segno le configurazioni dei bit sono semplicemente leggibili per l'uomo, non si può usare un sommatore convenzionale con notazione in modulo e segno

La notazione usata nei calcolatori è quella in complemento a 2 che consente al sommatore tradizionale di eseguire la somma algebrica fra due quantità. L'esempio in:

- blu 5 -4. Ris= **1**, Ov= 0, CO =1, Co<sub>n-2</sub>=1
- verde -2 + 4. Ris= **2**, Ov=0, CO = 1, Co<sub>n-2</sub>=1
- rosso -4 -5. Ris= **7**, Ov= 1, CO=1, Co<sub>n-2</sub>=0
- arancione 6+3. Ris= **-7**, Ov= 1, CO=0, Co<sub>n-2</sub>=1

**OVERFLOW** in complemento a 2!

**OVERFLOW: Errore ottenuto quando si ottiene un risultato al di fuori del range rappresentato.**

Si può avere overflow quando si sommano quantità dello stesso segno, se c'è overflow il risultato avrà segno opposto. Studiando bene a partire da questa affermazione e dalla struttura interna dell'adder si arriva alla formula  $Ov = CO \text{ XOR } Co_{n-2}$

# Considerazioni su somma: riporti e overflow

- Ipotesi: consideriamo qui  $n$  bit con  $n = 8$
- Se eseguiamola somma di quantità senza segno (i numeri rappresentabili vanno da 0 e 255) possiamo utilizzare 8 full adder e l'ultimo carry segnala con il valore 1 l'errore nel risultato che è troppo grande per essere rappresentato
- Se consideriamo la somma di numeri con segno e usiamo la notazione in complemento a 2, allora i numeri rappresentabili vanno da -128 a 127. In questo caso l'errore di **risultato non rappresentabile** non è indicato dal valore 1 del carry out dell'ultimo full adder. Occorre invece realizzare una rete logica che legga almeno i segni degli operandi, il segno dell'uscita ed il bit di carry out dell'ultimo full adder. In alternativa si ottiene la stessa informazione dallo XOR del carry in e del carry out dell'ultimo full adder. Questo XOR dà quindi l'indicazione della situazione di overflow. Questa soluzione, non dipende dal segno degli operandi.
- ATTENZIONE: l'ultimo bit di carry out verrà d'ora in poi chiamato “**carry out**” il termine overflow verrà utilizzato solo per indicare lo stato di errore dovuto a risultato non visualizzabile

# Esercizi

- Si dica quanti numeri interi si possono rappresentare con 4 bit in *Complemento a due* e quanti numeri si possono rappresentare con il codice *bit di segno e modulo*
- Si indichino tutti i numeri rappresentabili con 3 bit con i due codici *Complemento a due* e *bit di segno e modulo*, poi a ciascuno di questi numeri si associ la codifica effettuata secondo ciascuno dei due codici assegnati
- Utilizzando una rappresentazione in complemento a 2 con  $n=5$  si eseguano le seguenti operazioni:

$$(-12) + (+4)$$

$$(-12) - (-12)$$

$$(-12) + (-4)$$

$$(+10) - (+11)$$

$$(-12) + (-1)$$

$$(-12) + (+12)$$

$$(+10) - (+5)$$

$$(+12) + (+5)$$

# Esercizi

- **Se lavoro in complemento a 2 la situazione di overflow è determinata dal fatto che i CO calcolati dai full adder in posizione n-2 ed n-1 siano diversi?**

Caso della somma di operandi con segno opposto (che non porta mai a overflow)

1..... +

0.....

Se il risultato è negativo non c'è ovviamente stato riporto da n-2 e quindi neanche in n-1, se il risultato è positivo c'è stato quindi un riporto da n-2 e questo ha causato riporto in n-1, quindi i due CO in oggetto sono uguali

Caso della somma di operandi positivi (overflow solo se risultato negativo)

0..... +

0.....

Non è possibile avere riporto in n-1. Se il risultato è positivo non c'è riporto da n-2 se il risultato è negativo c'è stato un riporto da n-2, ma questo non può causare riporto anche in n-1 e quindi in questo caso i due CO sotto analisi sono diversi

Caso della somma di operandi negativi (overflow solo se risultato positivo)

1..... +

1.....

Abbiamo sicuramente riporto dalla posizione n-1, se il risultato è negativo c'è stato quindi per forza un riporto da n-2. Se il risultato è positivo vuol dire che non è arrivato riporto dalla posizione n-2 e quindi in questo caso i due CO sotto analisi sono diversi