Capitolo 2
Codifica dell'informazione

Reti Logiche T

2.1
Rappresentazione dell'informazione

Simbolo, alfabeto e stringa

Informazione - **Stringa** di lunghezza finita formata da **simboli** appartenenti ad un **alfabeto** di definizione A:

$s_1\ s_2 \ldots s_i \ldots s_{n-1}\ s_n$ con $s_i \in A : \{a_1, a_2, .., a_m\}$

Alfabeto A: insieme di informazioni "elementari"

Esempi di possibile coppie stringa/alfabeto:

"testo" e caratteri

"numero" e cifre

"immagine" e pixel/colore-toni di grigio

"parlato" e fonemi

"musica" e note

"disegno" e pendenza/lunghezza di tratti

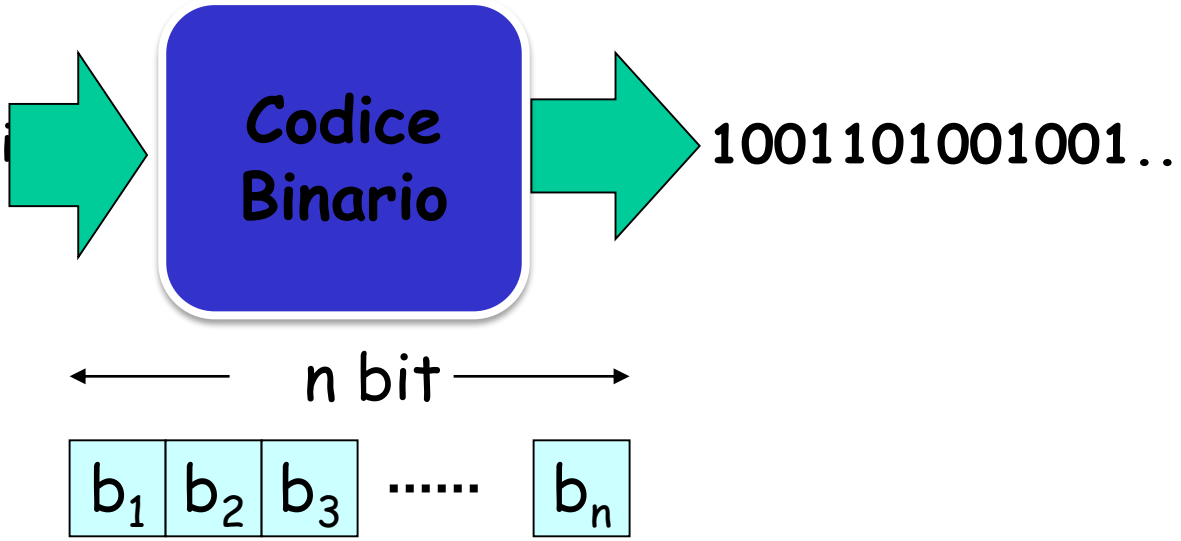
"misura" e posizione di un indice

...

La codifica binaria dell'informazione

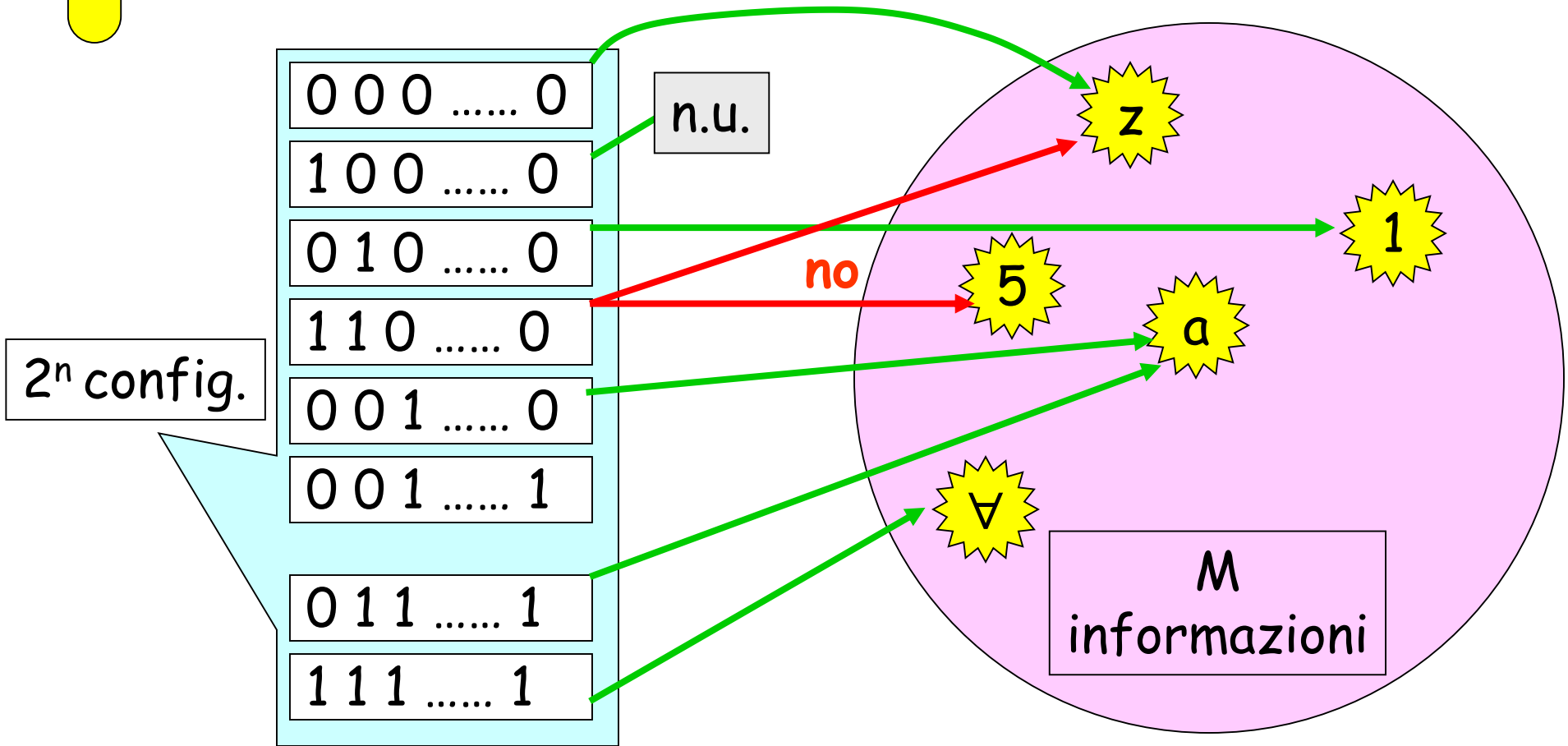
- L'utilizzo di **segnali binari** per l'elaborazione e la comunicazione di una macchina digitale comporta che
  - la macchina impieghi un **alfabeto binario** (ovvero che utilizzi i due soli simboli 0 e 1)
  - qualsiasi informazione elaborata o comunicata sia raprresentata da una stringa di bit
- Attraverso opportuni **codici binari** si trasformano dunque informazioni di varia natura in informazione sotto forma di stringhe di bit (**rappresentazione binaria dell'informazione**)
- Studieremo nel seguito tali modalità di rappresentazione

- Testi, immagini, suoni
- Istruzioni, dati, risultati
- Ingresso, uscita, stato
- ...



Codice binario

Codice binario - Funzione dall'insieme delle 2^n configurazioni di n bit ad un insieme di M informazioni (*simboli alfanumerici, colori, eventi, stati interni*, ecc.).
Condizione necessaria per la codifica: $2^n \geq M$
(se vi sono M simboli da codificare, occorrono almeno $2^n \geq M$ differenti configurazioni binarie)



Proprietà di un codice

Codice: rappresentazione convenzionale dell'informazione.

La scelta di un codice è condivisa da sorgente e destinazione, ed ha due gradi di libertà:

- il numero n di bit (qualsiasi, purché $2^n \geq M$)
- l'associazione tra configurazioni e informazioni

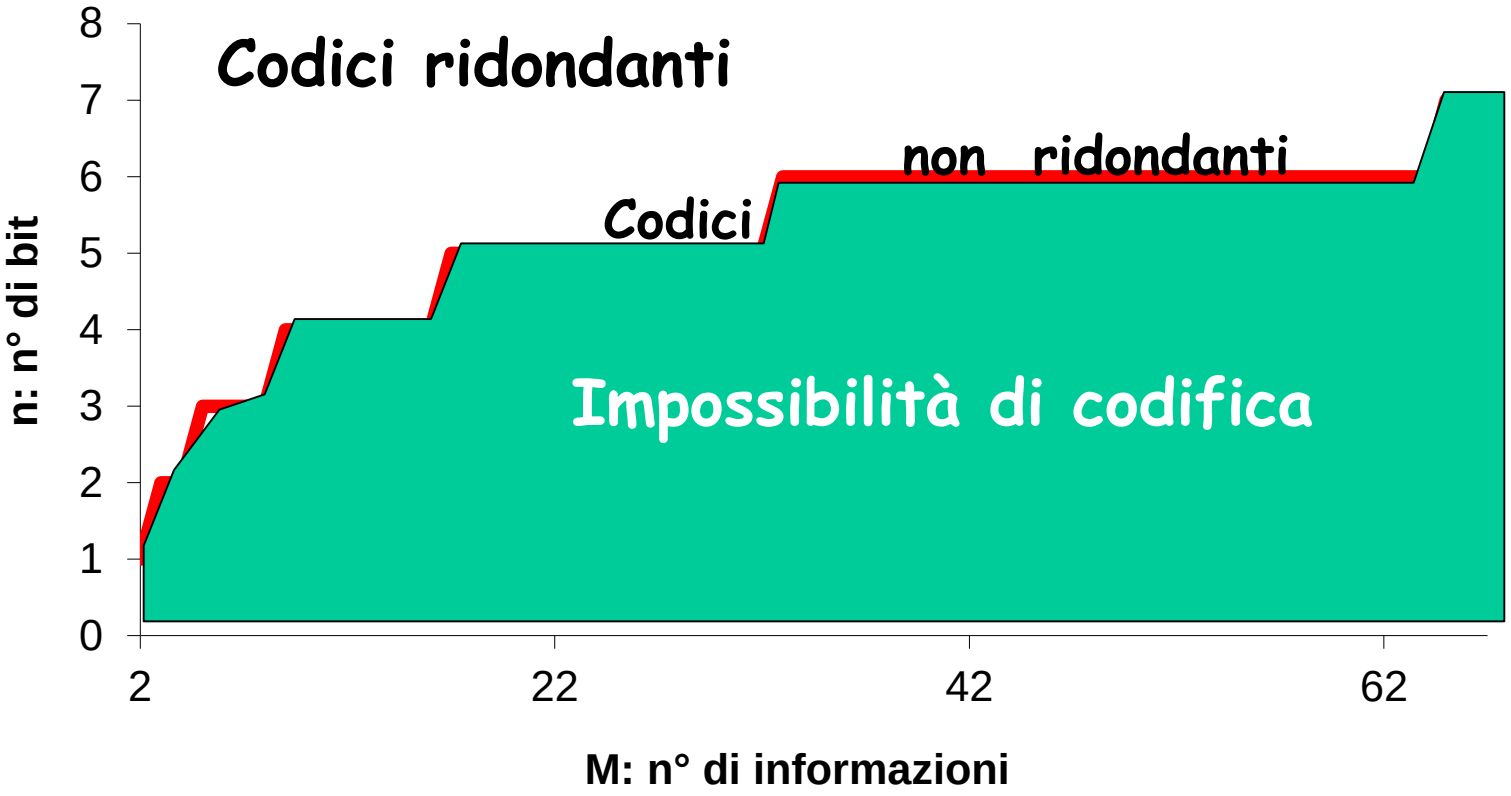
A parità di n e di M, le associazioni possibili sono:

$$C = 2^n! / (2^n - M)!$$

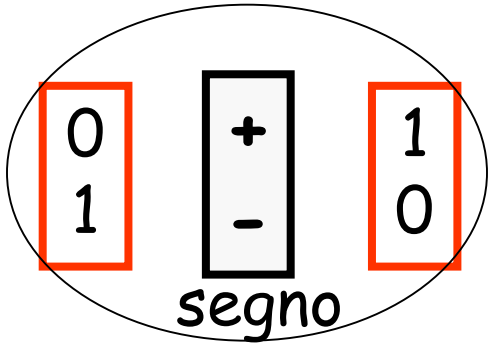
n.bit, n.informazioni	associazioni possibili
n = 1, M = 2	C = 2
n = 2, M = 4	C = 24
n = 3, M = 8	C = 40.320
n = 4, M = 10	C = 29.059.430.400

Codici ridondanti e codici non ridondanti

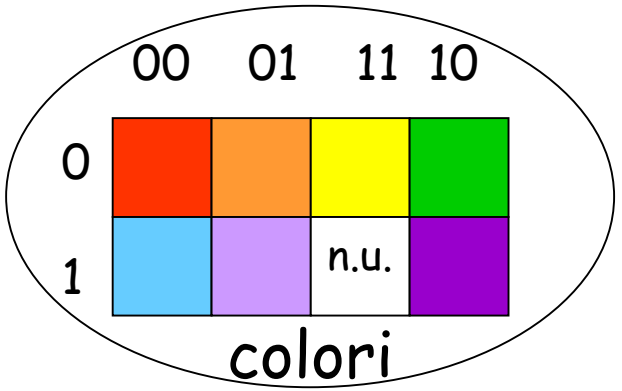
- Poichè $2^n \geq M$, dato M il n. minimo di bit è $n_{min} = \lceil lg_2\ M \rceil$
- Un codice che utilizza un numero $n > n_{min}$ è detto codice **ridondante**



M=2
n=1
C=2



M=8
n=3
C=40.320



cifre decimali

Non ridondanti			Ridondanti	
altri 29.059.430.399 codici a 4 bit	0000	*zero*	1111110	1000000000
	0001	*uno*	0110000	0100000000
	0010	*due*	1101101	0010000000
	0011	*tre*	1111001	0001000000
	0100	*quattro*	0110011	0000100000
	0101	*cinque*	1011011	0000010000
	0110	*sei*	0011111	0000001000
	0111	*sette*	1110000	0000000100
	1000	*otto*	1111111	0000000010
	1001	*nove*	1110011	0000000001
	BCD		7 segmenti	1/10

Binary-Coded Decimal

Per codificare un alfabeto di simboli formato dalle 10 cifre decimali (M=10) occorrono almeno n=4 bit. Ma è possibile utilizzarne anche 5, 6, ..

Codice BCD

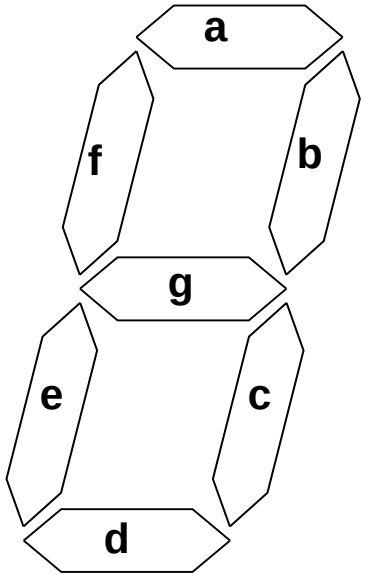
- Codice che rappresenta ciascuna cifra decimale con 4 bit
- k cifre decimali -> 4k bit

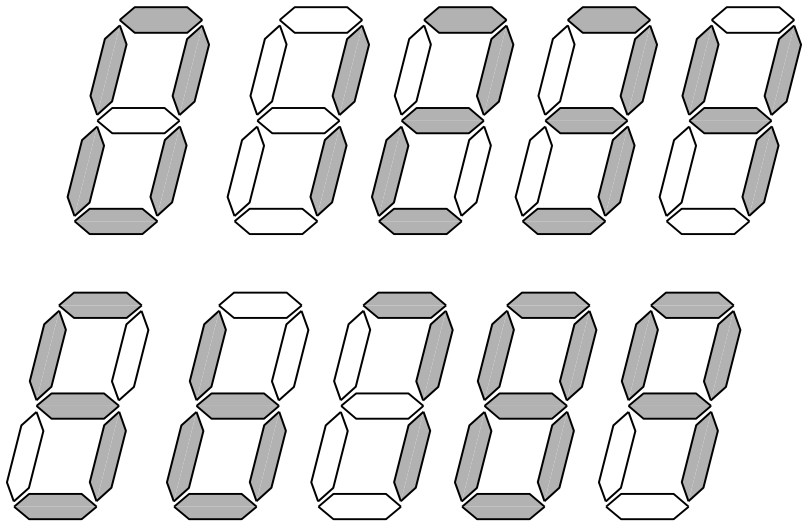
zero	0000
uno	0001
due	0010
tre	0011
quattro	0100
cinque	0101
sei	0110
sette	0111
otto	1000
nove	1001

- Es.: nel caso di due cifre decimali: ventuno -> 0010 0001

Codice a 7 segmenti

- Codice ridondante utilizzato per visualizzare a display numeri decimali
- M=10, n=7>4

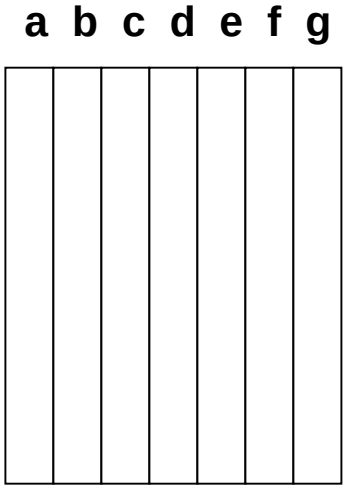


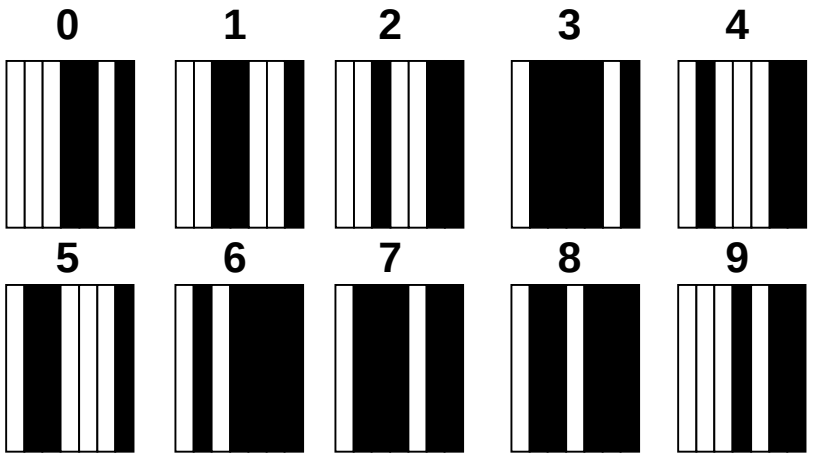


	a	b	c	d	e	f	g
zero	1	1	1	1	1	1	0
uno	0	1	1	0	0	0	0
ecc.							

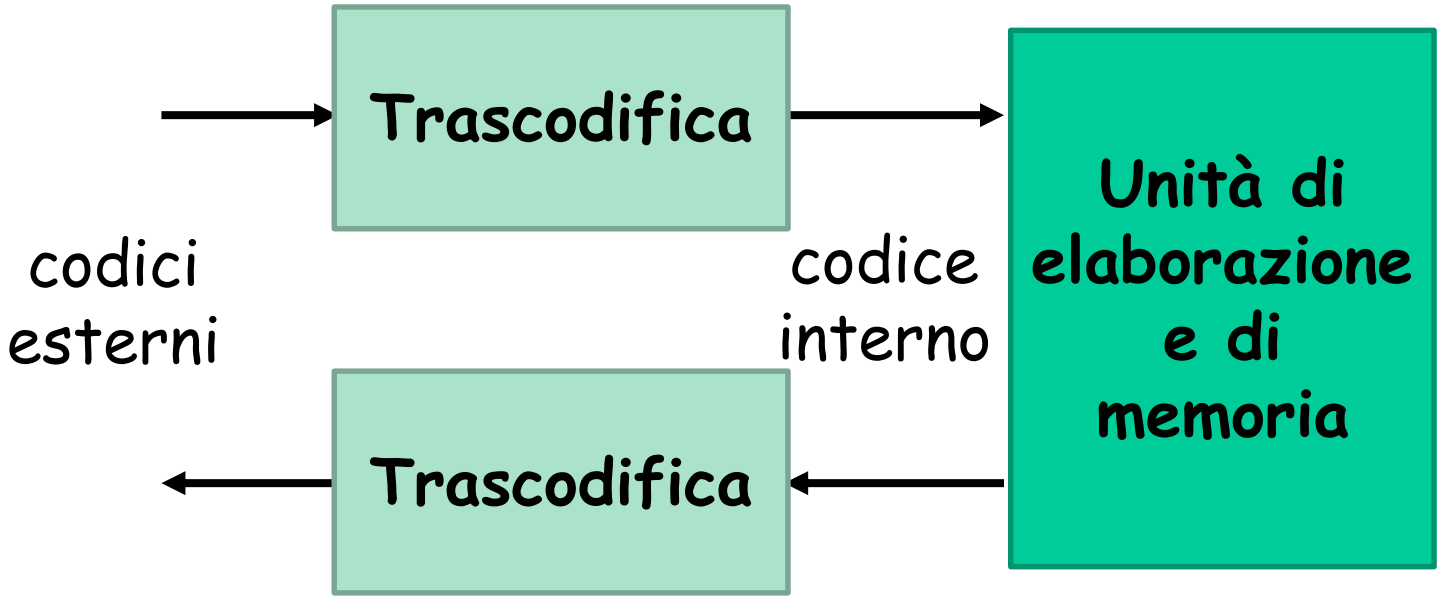
Universal Product Code

- Codice ridondante utilizzato per associare un identificativo (facilmente leggibile da un apposito sensore senza contatto) a un prodotto
- M=10, n=7>4





La conversione di codice (trascodifica)

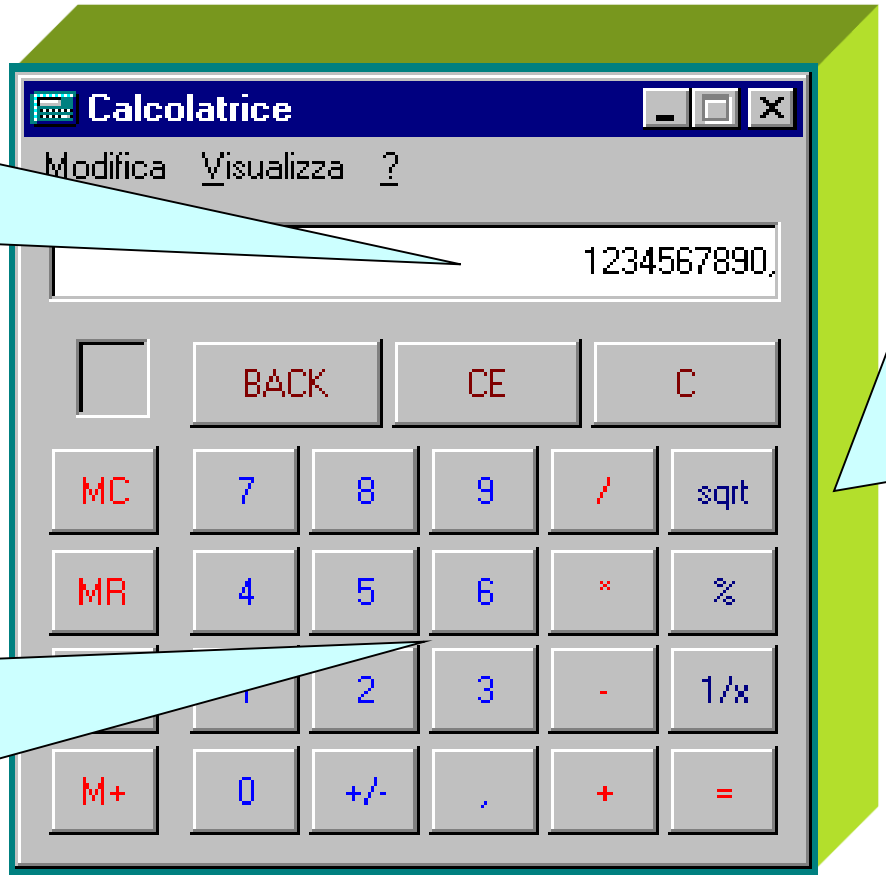


Il codice interno è di norma non ridondante per minimizzare il numero di bit da elaborare e da memorizzare.
Il codice esterno è di norma

- 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.

La calcolatrice tascabile

Codice ridondante per la visualizzazione dei dati

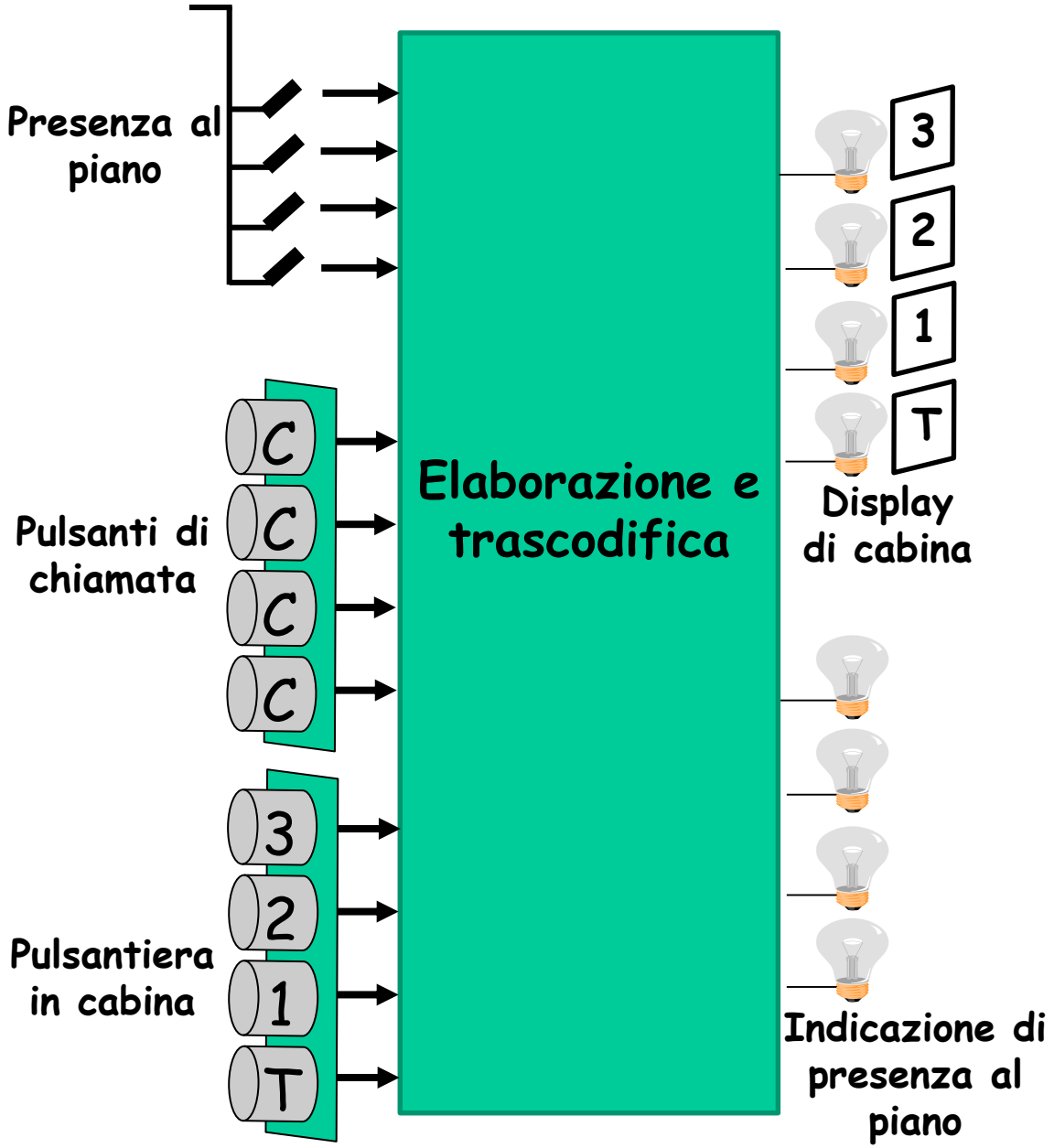


Codice ridondante per la introduzione dei dati e dei comandi

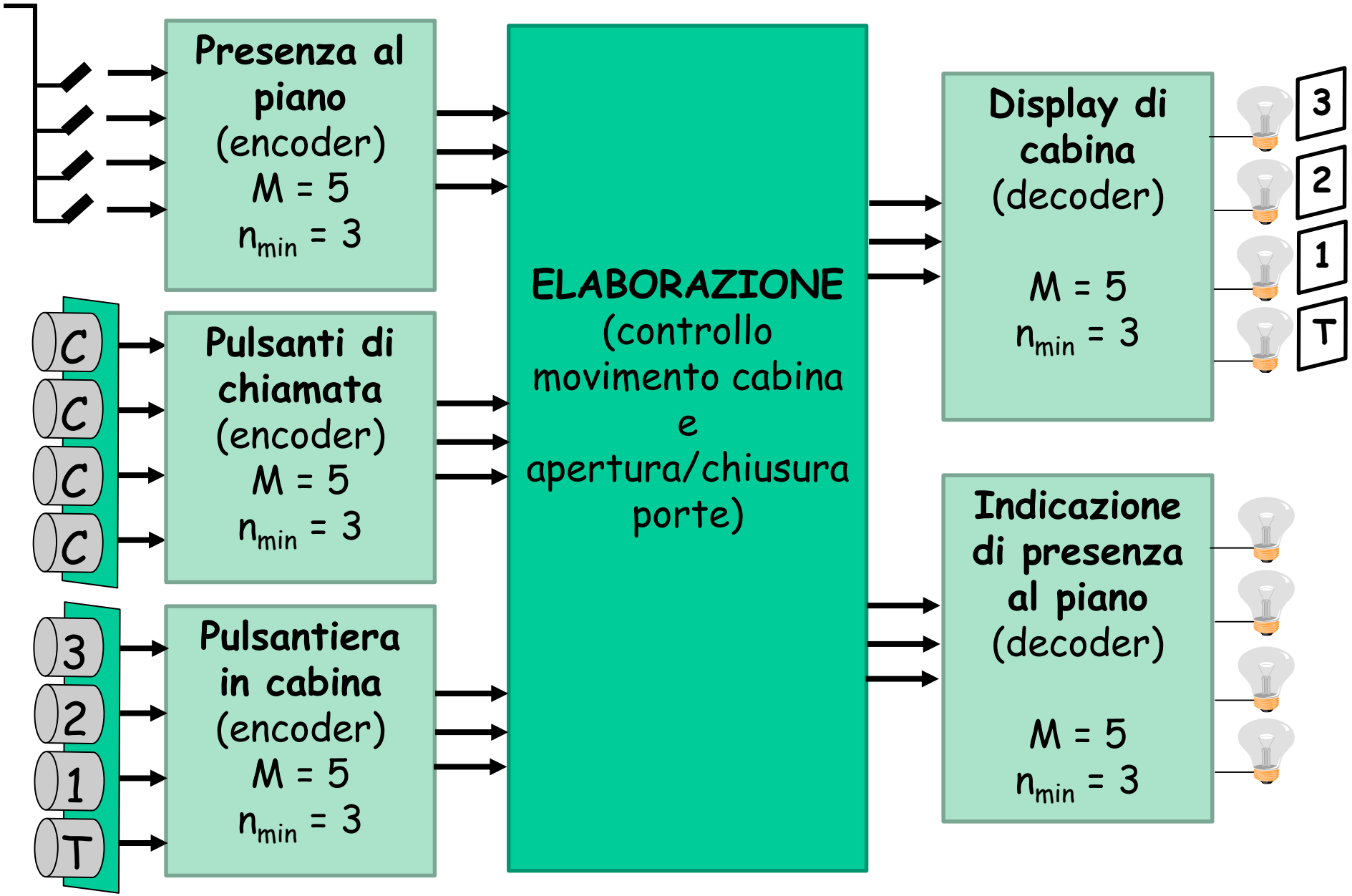
Codice BCD per la rappresentazione interna dei numeri

Esempio: l'ascensore

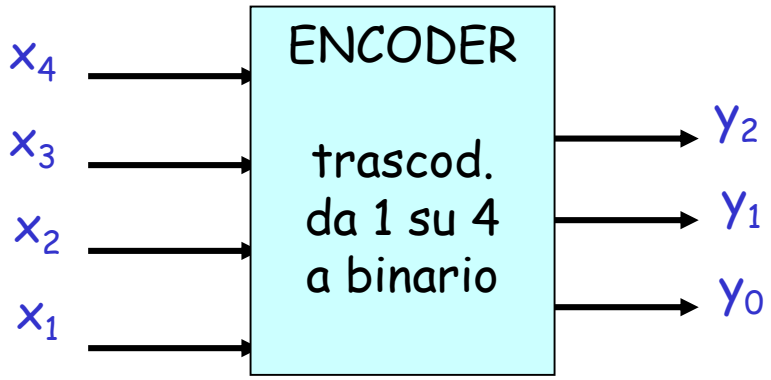
- Un ascensore serve 4 piani
- **Segnali in ingresso:**
  - presenza o meno della cabina a uno dei piani (4 rilevatori di presenza, 5 casi possibili)
  - richiesta di spostamento della cabina (pulsantiera a 4 tasti, 5 casi possibili)
  - in entrambi i casi: n=4, M=5, codice ridondante
- **Segnali in uscita:**
  - indicatore del piano corrente ai vari piani e in cabina (4 lampade, 5 casi possibili)
  - Anche in questo caso: n=4, M=5, codice ridondante



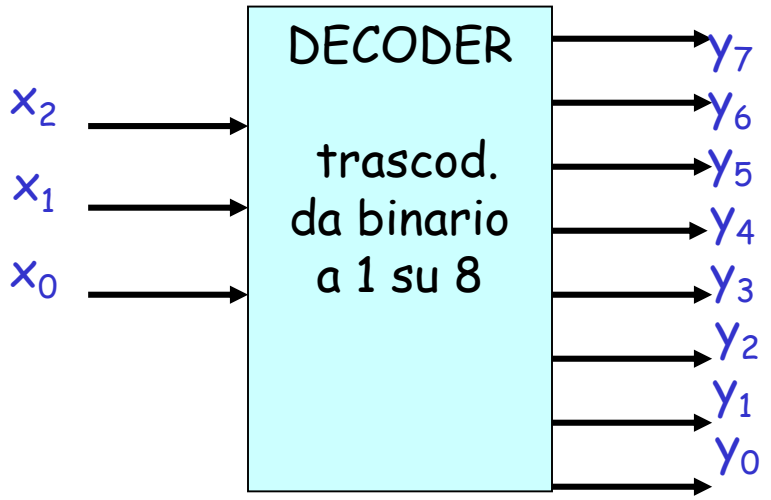
Input/output di un ascensore



Le due macchine combinatorie per la trascodifica nell'ascensore



x_4	x_3	x_2	x_1	y_2	y_1	y_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	1	1
1	0	0	0	1	0	0



↑ Codice «uno su N» ↓

x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Codici proprietari e codici standard

Codice proprietario - Codice scelto da un Costruttore per mettere in comunicazione macchine di sua produzione.
L'uso di codici proprietari mira ad ottimizzare le prestazioni e a proteggere il mercato di certe macchine.

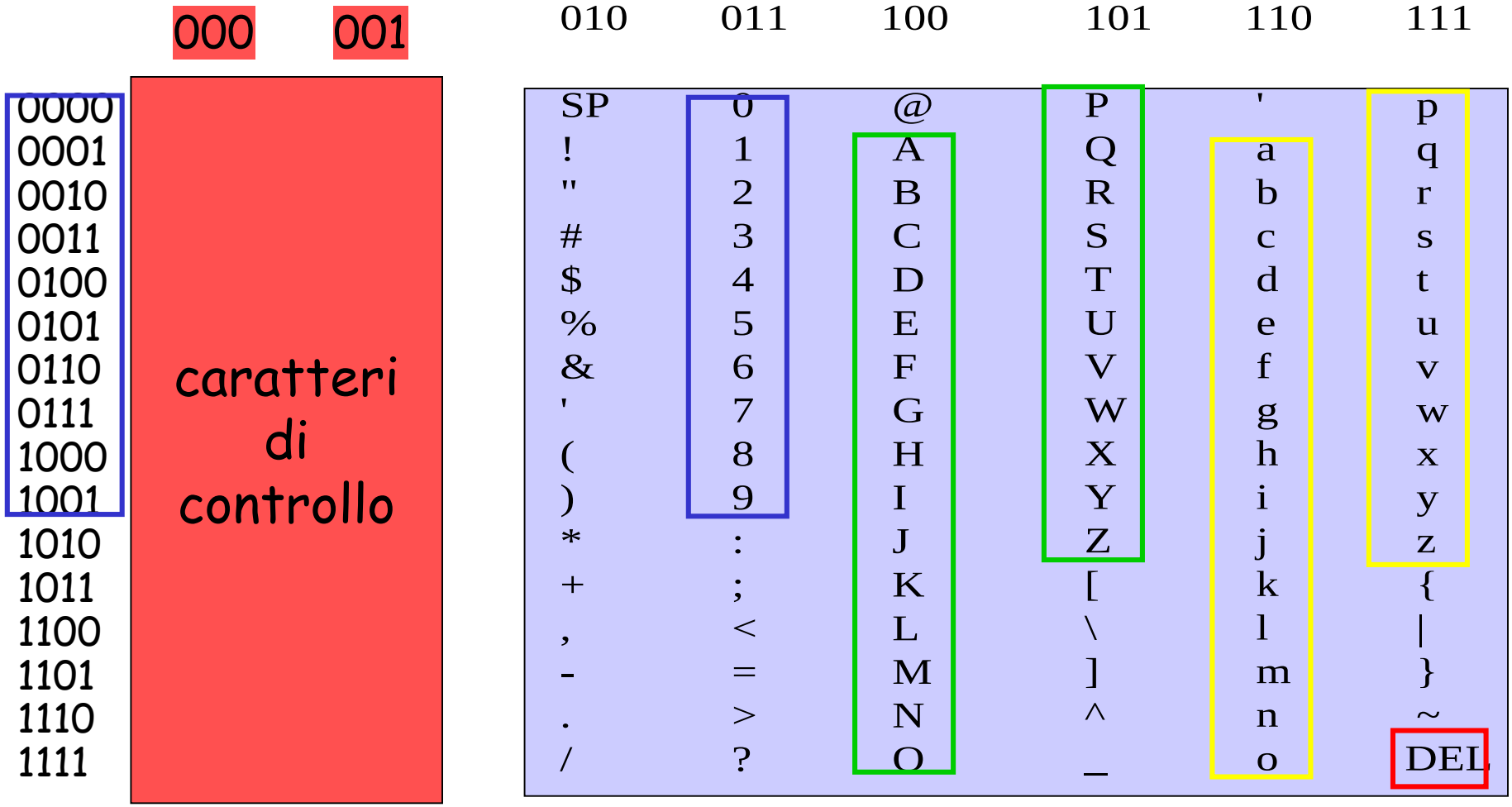
Esempi: Linguaggio Assembler, Telecomando TV

Codice standard - Codice scelto da norme internazionali (***de iure***) o dal Costruttore di una macchina ampiamente utilizzata sul mercato (***de facto***).
L'uso di codici standard nelle unità di I/O consente di collegare macchine realizzate da Costruttori diversi.

Esempi: Stampanti e Calcolatori, Calcolatori e Calcolatori

2.2 La codifica dei caratteri

Il codice ASCII a 7 bit (1967)



- 128 caratteri (33 di controllo)
- 0: 011 0000
- A: 100 0001; a: 110 0001
- DEL: 111 1111

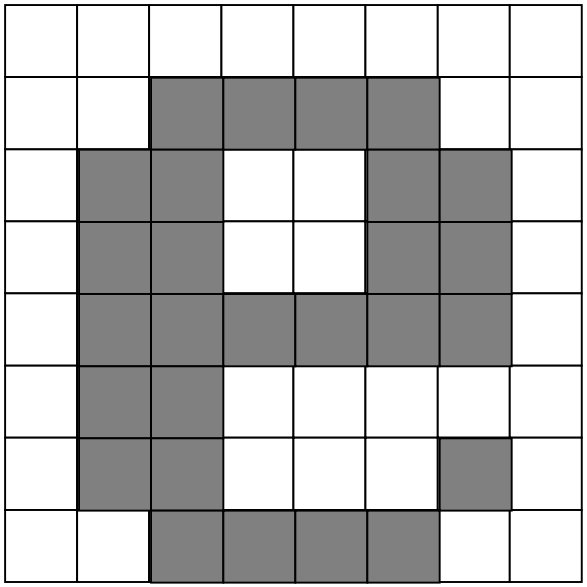
- **American Standard Code for Information Interchange**
- Primo standard de iure per la codifica binaria dei caratteri, per le comunicazioni a distanza di telescriventi

Codice ASCII esteso (8 e 16 bit)

- Estensione del codice ASCII a 8 bit, includendo nell'insieme dei caratteri rappresentati i simboli impiegati dalle lingue originate dal latino
- **Standard Unicode**: ulteriore estensione (**16 bit**) che permette di codificare in binario i simboli di tutte le lingue conosciute; mantiene la compabilità con ASCII

Bit map: un codice ridondante per simboli alfanumerici

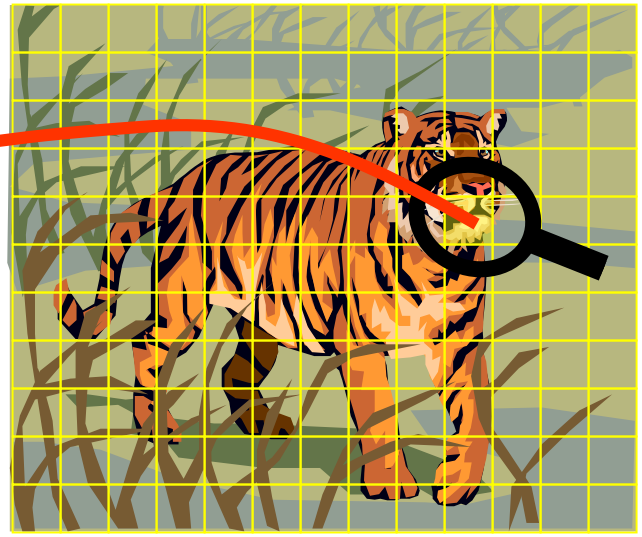
- Rappresentazione binaria dei simboli **grafici** dei caratteri
- Mentre nelle stampanti «ad impatto» basta il codice ASCII, per molti altri casi (getto d'inchiosto, laser, ma anche per mostrare a monitor) è necessario definire dei simboli grafici mediante matrici di pixel (***bitmap***)
- Ciascun elemento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

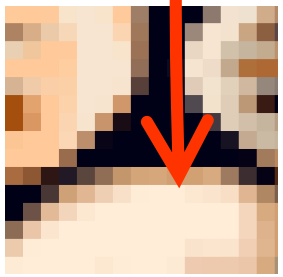


Es. di bitmap 8x8

Codifica di immagini

Scena reale





R ∈ {0,1,2,..,254,255}
G ∈ {0,1,2,..,254,255}
B ∈ {0,1,2,..,254,255}

24 bit/pixel

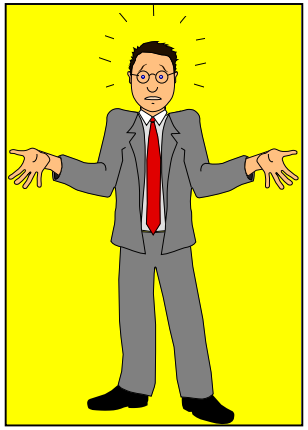
Immagine digitalizzata: 720 * 480 * 24 = 8.294.400 ≅ 8 Mbits

Codifica di immagini video

In tale ottica, la registrazione di un film della durata di 2 ore comporta una capacità di memoria pari a (30 frame/s):

2 * 60 * 60 * 30 * 8 Mbits = 1728 Gbits

Essendo la capacità di un DVD pari a 37.6 Gbits,
è sufficiente (!?) allo scopo dotarsi di 46 DVD.
A meno che …



Tecniche di compressione

Tipologie di compressione:

- con perdita di informazione (**lossy**, es. JPEG, MP3, MPEG)
- senza perdita di informazione (**lossless**, es. Lempel-Ziv)

Contesto tipico, rispettivamente:

- elaborazione audio e video
- elaborazione di testi

Efficienza o fattore di compressione:

$$\frac{\text{Size [Original (Uncompressed) Info]}}{\text{Size [Compressed Info]}}$$

2.3 La codifica dei numeri

Rappresentazione dei numeri

- Consideriamo il problema di dover rappresentare l'insieme degli $M = 10^k$ numeri interi non negativi: $\{0,1,\ldots,10^k-1\}$
  - Es. : con k=1: {0,1,…,9} ; con k=2: {0,1,…,99}; …
- Tali numeri sono composti da k cifre decimali (1 cifra per M=10, 2 cifre per M=100, e così via)
- Il numero minimo di bit (n_{min}) necessario per rappresentarli è:

$$n_{min} = \lceil \lg_2 M \rceil = \lceil \lg_2 10^k \rceil = \lceil k * \lg_2 10 \rceil = \lceil k * 3{,}32 \rceil.$$

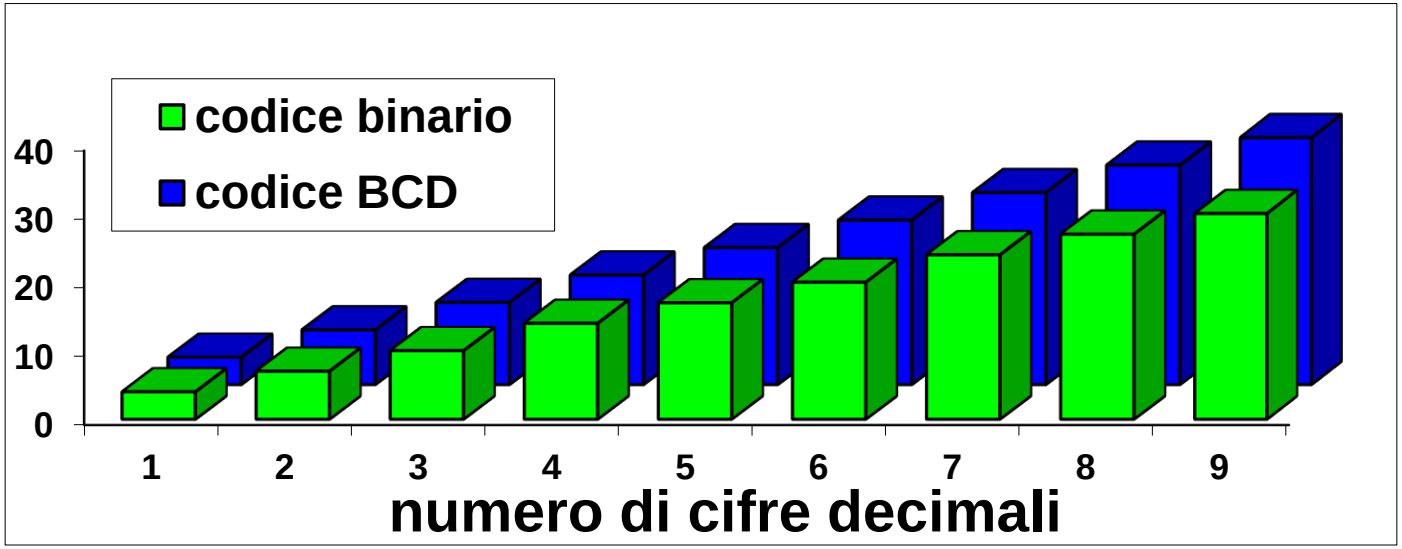
- Il codice BCD, ancorché irridondante dal punto di vista della rappresentazione delle singole cifre decimali (ovvero quando **k=1**), comporta un numero di bit decisamente maggiore con k>1, in quanto utilizza sempre 4 bit per ciascuna cifra decimale:

$$n_{BCD} = k * \lceil \lg_2 10 \rceil = k * \lceil 3{,}32 \rceil = k * 4 > n_{min},\ k = 2, 3, \ldots .$$

- Es.:
  - con k=1: $n_{min} = \lceil 1 * 3{,}32 \rceil = 4 = n_{BCD}$
  - con k=2: $n_{min} = \lceil 2 * 3{,}32 \rceil = 7 < n_{BCD} = 8$

Rappresentazione dei numeri

- La ridondanza è tanto più significativa quanto più elevato è il valore di k.



n_{BCD}	4	8	12	16	20	24	28	32	36
n_{min}	4	7	10	14	17	20	24	27	30
$n_{BCD} - n_{min}$	0	1	2	2	3	4	4	5	6

Rappresentazione dei numeri

- Il codice BCD (assieme a ASCII, Unicode, etc..) viene dunque utilizzato per codificare i numeri esternamente alla macchina digitale

- Per quanto riguarda la rappresentazione dei numeri interna si utilizza il sistema di numerazione in base 2
  - minima occupazione di memoria
  - massima velocità d'esecuzione di operazioni (somme, sottrazioni, moltiplicazioni, ..)

Sistemi di numerazione

Un sistema di numerazione è definito da:

- un **insieme di simboli elementari**;
- un **insieme di regole** che stabiliscono le modalità **di rappresentazione** di grandezze numeriche in termini di simboli elementari;
- un **insieme di regole** che stabiliscono le modalità **di elaborazione** di grandezze numeriche espresse in notazione simbolica.

I sistemi di numerazione si distinguono in:

- **sistemi non posizionali** (es. sistema di numerazione romano)
- **sistemi posizionali** (es. sistema di numerazione decimale).

1667 **MDCLXVII**

Sistema di numerazione posizionale in base b (b≥2)

- A partire da **b** (*base del sistema di numerazione*) e da **n+m** coefficienti che rappresentano il numero in tale base, è possibile ottenerne il relativo valore mediante polinomio in cui ciascuna cifra è pesata da una diversa potenza di b

1) Rappresentazione:

$$(N_b) = (a_{n-1} \dots a_0, a_{-1} \dots a_{-m})_b \qquad a_k \in \{0, 1, \dots, b-1\}, \ \forall \ k$$

2) Valore:

$$(N_b) = (a_{n-1}\, b^{n-1} + \dots + a_0\, b^0 + a_{-1}\, b^{-1} + \dots + a_{-m}\, b^{-m})_b$$

I sistemi di numerazione binario, ottale, esadecimale

base=2, {0,1}

base=8, {0,1,2,3,4,5,6,7}

base=16, {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Conversione di rappresentazione (base) tra base 2,8 e 16

B → B' (B, B' = 2, 8, 16):

B = 2 → B' = 8, 16

N_2 = 101011000100

101-011-000-100 → N_8 = 5304

1010-1100-0100 → N_{16} = *AC4*

B = 8, 16 → B' = 2

N_8 = 5236 → N_2 = 101010011110

N_{16} = E82 → N_2 = 111010000010

N_{10}	N_2	N_8	N_{16}
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
...	...	...	...

Conversione di base

$$(N)_B = (I,F)_B = (a_{n-1} \dots a_0, a_{-1} \dots a_{-m})_B \rightarrow (N)_{B'} = (?,?)_{B'}$$

I: parte intera; F: parte frazionaria. Es. $(131,75)_{10}$

- Problema: conversione della rappresentazione di un numero da base B a base B'

- **Metodo di conversione polinomiale**:
  - si avvale delle regole di rappresentazione e di elaborazione del sistema di numerazione in base B'
  - pertanto è convenientemente utilizzabile per operare la conversione dalla base B = $\forall$ (di norma 2) alla base B'= 10.
- **Metodo di conversione iterativa:**
  - si avvale delle regole di rappresentazione e di elaborazione del sistema di numerazione in base B
  - pertanto è convenientemente utilizzabile per operare la conversione dalla base B = 10 alla base B' = $\forall$ (di norma 2).

Metodo di conversione polinomiale

- Da utilizzare per passare da B a 10
- Il metodo di conversione polinomiale sfrutta la rappresentazione polinomiale in n+m coefficienti di un numero:

$$(N)_B = (a_{n-1} \dots a_0, a_{-1} \dots a_{-m})_B$$

- Il numero nella rappresentazione della base di destinazione B' si ottiene, a partire dalla rappresentazione in base B, come:

$$(N)_{B'} = (a_{n-1} B^{n-1} + \dots + a_0 B^0 + a_{-1} B^{-1} + \dots + a_{-m} B^{-m})_{B'}$$

Esempio: $B = 2 \rightarrow B' = 10$

$$(100110{,}01)_2 = (0 \cdot 1 + 1 \cdot 2 + 1 \cdot 4 + 0 \cdot 8 + 0 \cdot 16 + 1 \cdot 32 + 0 \cdot 0{,}5 + 1 \cdot 0{,}25)_{10} = (38{,}25)_{10}$$

Esempio: $B = 3 \rightarrow B' = 10$

$$(122)_3 = (2 \cdot 1 + 2 \cdot 3 + 1 \cdot 9)_{10} = (17)_{10}$$

Metodo di conversione iterativa

- Da utilizzare per passare da B=10 a B'
- $c_{i-1} \dots c_0, c_{-1} \dots c_{-f}$ sono i coefficienti incogniti nella base B' di destinazione

- Procedimento per la **parte intera (I)** di un numero N:
  - c_0 si può determinare come resto (parte frazionaria) della divisione intera di I per B'
  - Iterando, ottengo i rimanenti coefficienti della parte intera ($c_{i-1} \dots c_1$)

$$(N)_{B'} = (c_{i-1} \dots c_0, c_{-1} \dots c_{-f})_{B'}$$

$$\begin{aligned}(I)_B &= (c_{i-1} B'^{i-1} + \dots + c_1 B' + c_0)_B \\ &= ((c_{i-1} B'^{i-2} + \dots + c_1) B' + c_0)_B \\ &= (I' B' + c_0)_B \end{aligned}$$

$\rightarrow$ c_0 = parte frazionaria di $(I/B')_B$,
c_1 = parte frazionaria di $(I'/B')_B$,
…

Metodo di conversione iterativa

- Procedimento duale si utilizza per i coefficienti della parte frazionaria (F)
- ciascun coefficiente è la parte intera della moltiplicazione di F per B'

$$
\begin{aligned}
(F)_B &= (c_{-1}\, B'^{-1} + c_{-2}\, B'^{-2} + \ldots + c_{-f}\, B'^{-f})_B \\
&= ((c_{-1} + c_{-2}\, B'^{-1} + \ldots + c_{-f}\, B'^{-f+1})\, B'^{-1})_B \\
&= ((c_{-1} + F')\, B'^{-1})_B \\
&\rightarrow c_{-1} = \text{parte intera di } (F \cdot B')_B, \\
&\quad\; c_{-2} = \text{parte intera di } (F' \cdot B')_B, \\
&\quad\; \ldots
\end{aligned}
$$

Metodo di conversione iterativa

Esempio: B = 10 → B' = 2

$(131{,}75)_{10} = (10000011{,}11)_2$

I	: 2 =	I'	+	c_k (k = 0, 1, …)
131		65		1
65		32		1
32		16		0
16		8		0
8		4		0
4		2		0
2		1		0
1		0		1

F	· 2 =	c_{-k} (k=1,2,…)	+	F'
0,75		1		0,5
0,5		1		0

Metodo di conversione iterativa

Al contrario di quanto avviene per la parte intera, il procedimento di conversione della parte frazionaria può non terminare in un numero finito di iterazioni. In tal caso la conversione è da intendersi completata al raggiungimento della precisione desiderata, con un eventuale arrotondamento dell'ultima cifra significativa.

Esempio: $B = 10 \rightarrow B' = 2$

$(\ldots,8)_{10} = (\ldots\ldots,11001101)_2$

F	$\cdot 2 =$	$c_{-k\ (k=1,2,\ldots)}$	$+$	F'
0,8		1		0,6
0,6		1		0,2
0,2		0		0,4
0,4		0		0,8
0,8		...		...

E se $B, B' \neq 10$? $B \rightarrow 10 \rightarrow B'$

Esempio: $B = 3 \rightarrow B' = 16$

$(122)_3 = (17)_{10} = (11)_{16}$

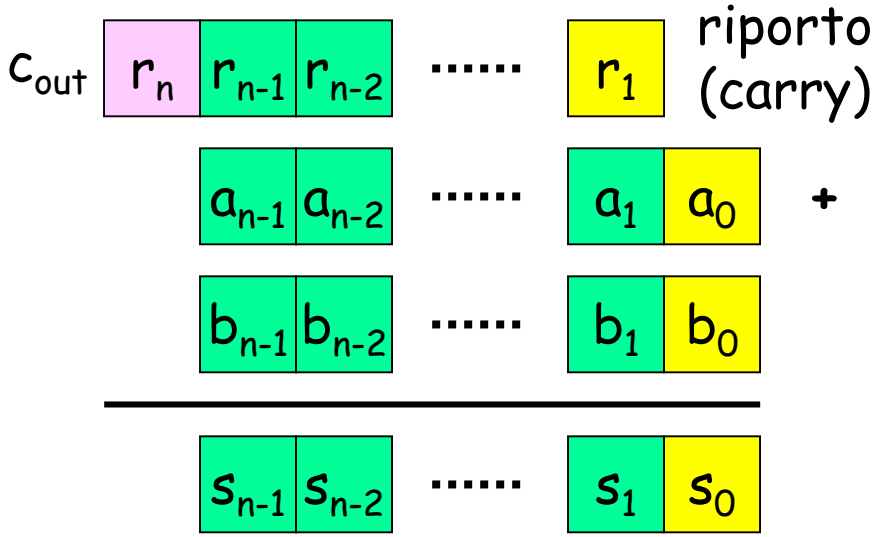
Addizione …

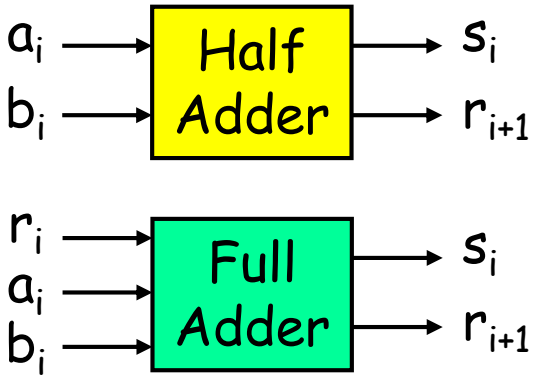
- Addizione tra due bit genera un risultato di 2 bit: **somma** (minor peso) e **riporto** (maggior peso)

A B	r s
0+0 =	0 0
0+1 =	0 1
1+0 =	0 1
1+1 =	1 0

r_i	a_i	b_i	r_{i+1}	s_i
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

- L'addizione di due numeri a n-bit genera un risultato di **n+1** bit
- S = A + B, con
  - A, B, S: n-bit unsigned integer
  - $0 \leq A, B, S \leq 2^n-1$





… Addizione …

- **Esempi con n=4:**
- Nel caso di n= 4, il massimo valore rappresentabile dalla somma S è pari a $2^4-1 = 15$

$C_{out} = 0$

$\downarrow$

$S \leq 2^4 - 1$

0	0	0	1			
	0	1	0	1	A	$(5)_{10}$
	1	0	0	1	B	$(9)_{10}$
	1	1	1	0	S	$(14)_{10}$

$C_{out} = 1$

$\downarrow$

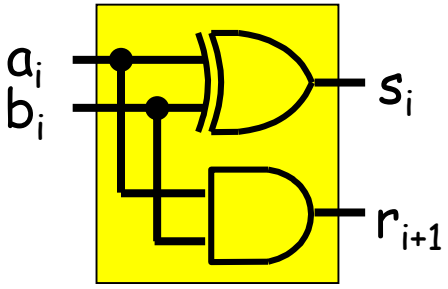
$S > 2^4 - 1$

1	0	0	0			
	1	0	0	0	A	$(8)_{10}$
	1	0	0	1	B	$(9)_{10}$
	0	0	0	1	S	$(17)_{10}$

- Il valore di c_{out} indica dunque se la somma è rappresentabile con n bit

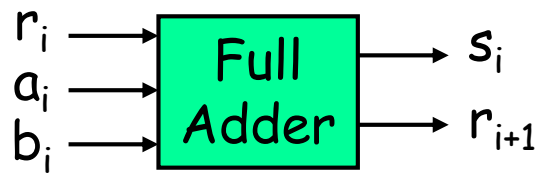
… Addizione …

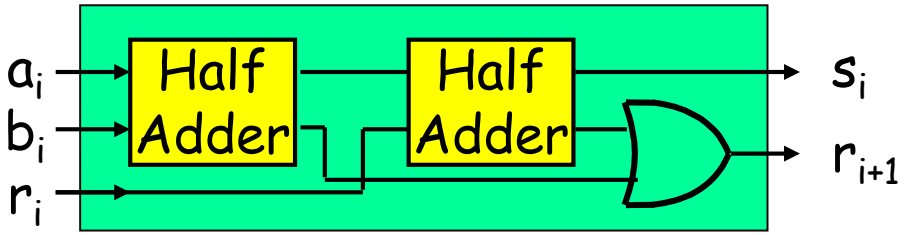
- Un Half-Adder è facilmente realizzabile mediante un EX-OR (somma) e un AND (riporto)

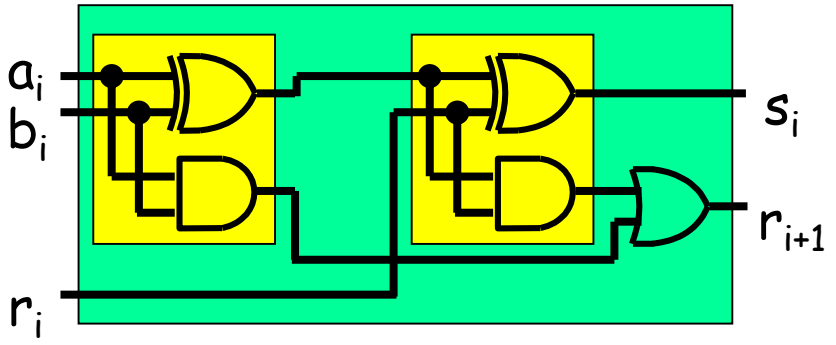


r_i	a_i	b_i	r_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Analogamente, un Full-Adder è realizzabile a partire da due Half-Adder (lo dimostreremo in seguito, quando parleremo di manipolazione algebrica)

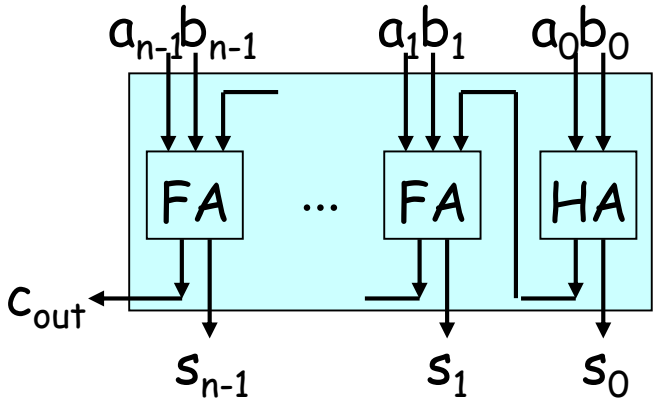




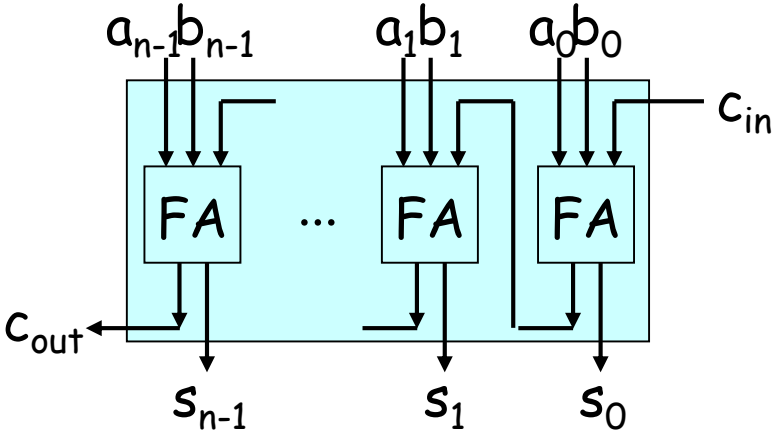


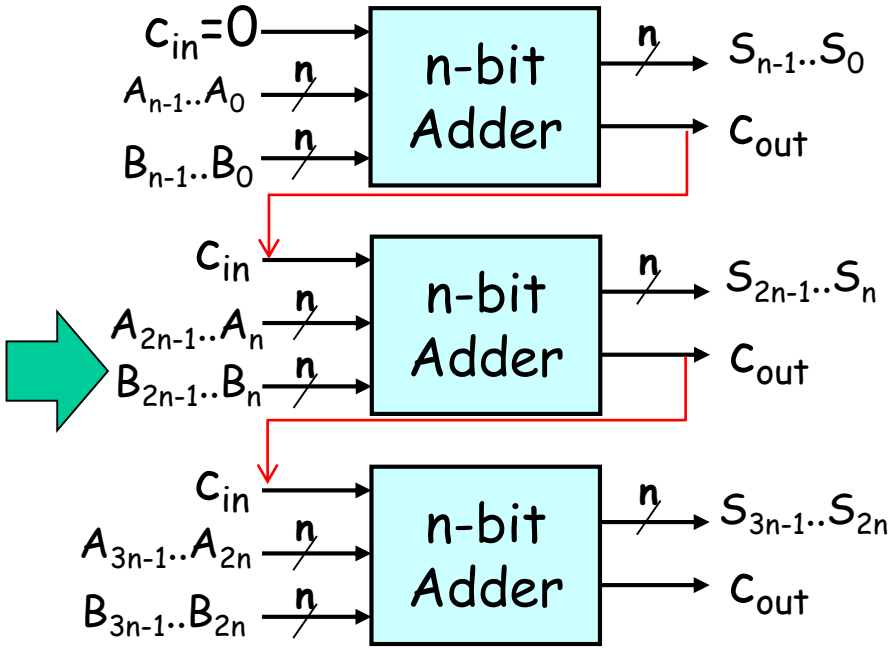
… Addizione …

- è possibile realizzare un sommatore a n bit componendo in serie n-1 Full Adder e un Half Adder



- Sostituendo il singolo HA con un FA si ottiene un componente con l'ingresso aggiuntivo di carry-in, utile per realizzare a sua volta FA con numero > n di ingressi (*modularità*)





(Sommatore a 3n bit)

Rappresentazione dei numeri relativi

- La rappresentazione di un numero con segno non è univoca
- In entrambe le rappresentazioni descritte il bit più significativo indica il segno

$A = a_{n-1}\ a_{n-2} \ldots a_1\ a_0$ n-bit signed integer

Segno e valore assoluto

a_{n-1} (0:+, 1:-) $a_{n-2} \ldots a_1\ a_0$

$-(2^{n-1}-1) \le A \le 2^{n-1}-1$

n = 4

$(+5)_{10} \rightarrow A = 0101$

$(-5)_{10} \rightarrow A = 1101$

Complemento a 2

$^2(-A) = 2^n - {}^2(A) = \text{not } {}^2(A) + 1$

$-2^{n-1} \le A \le 2^{n-1}-1$

n = 4

$(+5)_{10} \rightarrow A = 0101$

$(-5)_{10} \rightarrow A = 1010 + 1 = 1011$

- Proprietà rappr. in complemento a 2:
  - N ≥0: equivale a segno-v.a.
  - N<0: $^2(-N)$, con N espresso in segno-v.a.

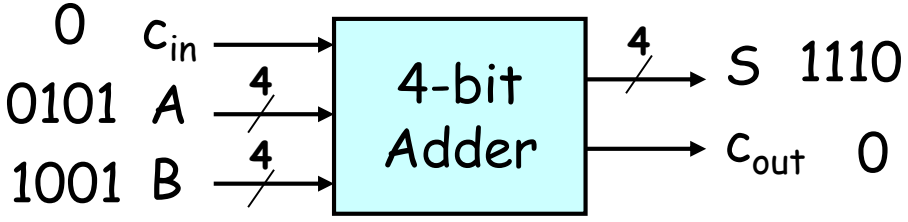
n = 4

	a_3	a_2	a_1	a_0	
+7	0	1	1	1	+7
+6	0	1	1	0	+6
+5	0	1	0	1	+5
+4	0	1	0	0	+4
+3	0	0	1	1	+3
+2	0	0	1	0	+2
+1	0	0	0	1	+1
+0	0	0	0	0	+0
-7	1	1	1	1	-1
-6	1	1	1	0	-2
-5	1	1	0	1	-3
-4	1	1	0	0	-4
-3	1	0	1	1	-5
-2	1	0	1	0	-6
-1	1	0	0	1	-7
-0	1	0	0	0	-8

Addizione di numeri relativi

- E' possibile usare il medesimo addizionatore per i numeri con e senza segno?

- Esempio con A=0101, B=1001, c_{in}= 0
- Il segnale in uscita all'adder è 1110 (valore diverso a seconda della rappresentazione!)



A, B: 4-bit unsigned integers

$A = (5)_{10}$ $B = (9)_{10}$ $S = (14)_{10} = A + B$ **OK**

A, B: 4-bit signed integers

Complemento a 2

$A = (+5)_{10}$ $B = (-7)_{10}$ $S = (-2)_{10} = A + B$ **OK**

Segno e valore assoluto

$A = (+5)_{10}$ $B = (-1)_{10}$ $S = (-6)_{10} \neq A + B = (+4)_{10}$ **NO**

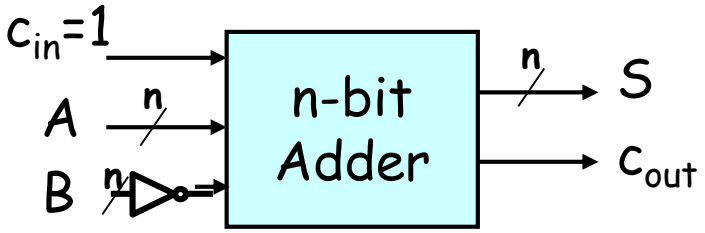
- La rappresentazione in complemento a 2 (al contrario di quella con segno e valore assoluto) ci permette di utilizzare il **medesimo componente** utilizzato per sommare numeri senza segno anche per sommare numeri relativi

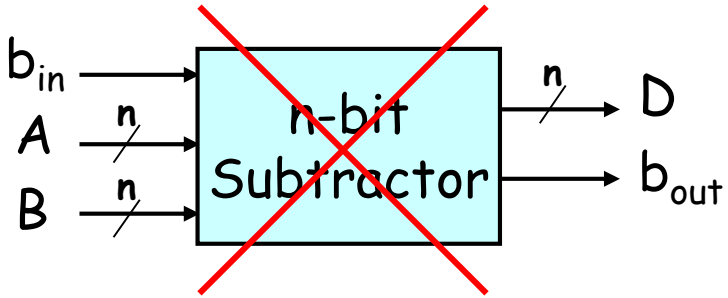
Sottrazione di numeri relativi

A, B: n-bit 2'complement signed integers

L'operazione di sottrazione è riconducibile ad un'operazione di addizione, previa **complementazione del sottraendo (e c_{in}=1):**

$$A - B = A + {}^{2}(-B) = A + \text{not}(B) + 1$$





n = 4

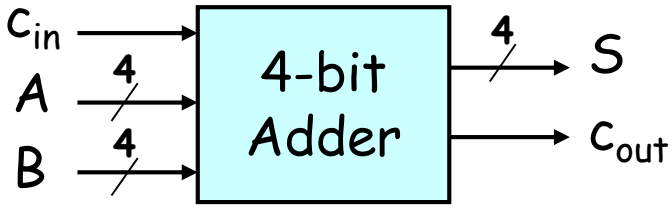
$A = (+5)_{10}$ $B = (+2)_{10}$ $A - B = (+5 + (-2))_{10} = (+3)_{10}$

0101 0010 0101 + (-0010) = 0101 + 1101 + 1 = 1 0011

??? OK

- è dunque possibile utilizzare lo stesso componente (Adder a n-bit) per:
  - sommare numeri con e senza segno (v. slide precedente)
  - sottrarre numeri con segno
- Il carry-out del FA, nel caso delle operazioni con numeri con segno, non è più indicativo della non rappresentabilità del risultato

Carry-out & Overflow



- c_{out} evidenzia correttamente la non rappresentabilità del risultato mediante n bit solo nel caso di **unsigned** integers

- L'analoga indicazione nel caso di **signed** integers è derivabile dal confronto del segno degli operandi e del risultato:

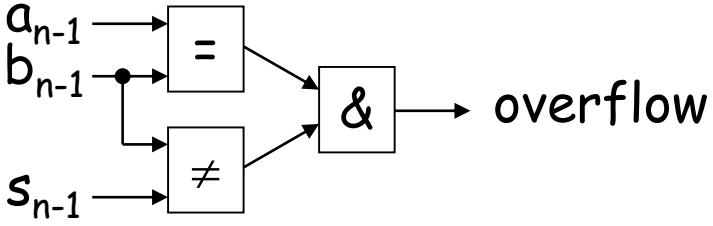
s_{n-1} \ a_{n-1} b_{n-1}	00	01	10	11
0	OK	OK	OK	NO
1	NO	OK	OK	OK

A, B, S: 4-bit

unsigned / signed integers

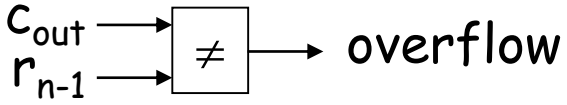
OK	OK	0	0	0	0			
$(6)_{10}$	$(+6)_{10}$		0	1	1	0	A	
$(1)_{10}$	$(+1)_{10}$		0	0	0	1	B	
$(7)_{10}$	$(+7)_{10}$		0	1	1	1	S	
OK	NO	0	1	1	1			
$(7)_{10}$	$(+7)_{10}$		0	1	1	1	A	
$(1)_{10}$	$(+1)_{10}$		0	0	0	1	B	
$(8)_{10}$	$(-8)_{10}$		1	0	0	0	S	
OK	OK	1	0	0	1			
$(9)_{10}$	$(-7)_{10}$		1	0	0	1	A	
$(9)_{10}$	$(-7)_{10}$		1	0	0	1	B	
$(2)_{10}$	$(+2)_{10}$		0	0	1	0	S	
OK	NO	1	1	1	1			
$(15)_{10}$	$(-1)_{10}$		1	1	1	1	A	
$(1)_{10}$	$(+1)_{10}$		0	0	0	1	B	
$(0)_{10}$	$(+0)_{10}$		0	0	0	0	S	

Signed & Unsigned Integers: +/-

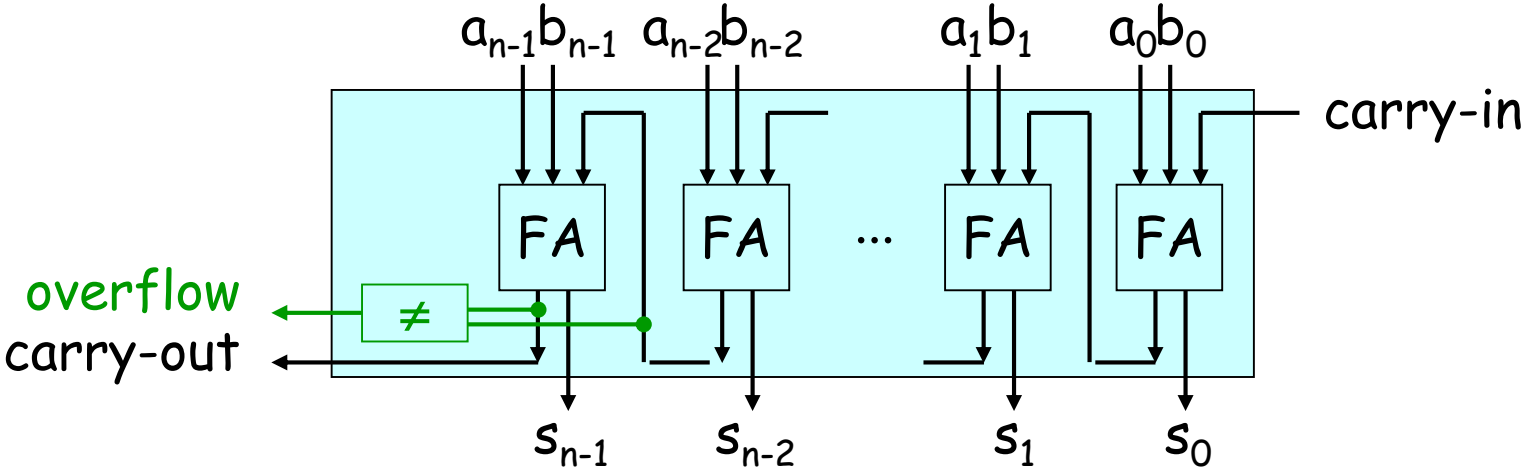


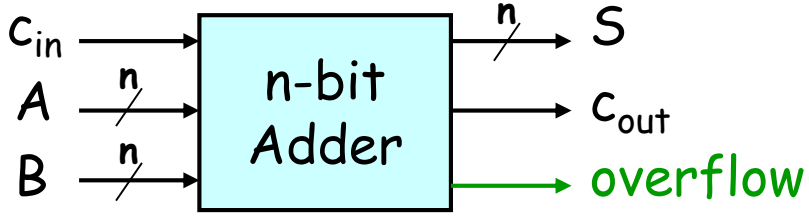
il segno dei due operandi è identico, e diverso da quello della somma

o più semplicemente



il carry-out dell'ultimo stadio è diverso dal carry-in dell'ultimo stadio





unsigned integers

signed integers

Sottrazione di numeri senza segno

- Lo stesso componente (n-bit adder) può essere usato anche per sottrazioni tra numeri senza segno?
- Complemento a 2 il sottraendo, interpreto il risultato come numero senza segno

	Add	Sub
Signed	✓	✓
Unsigned	✓	?

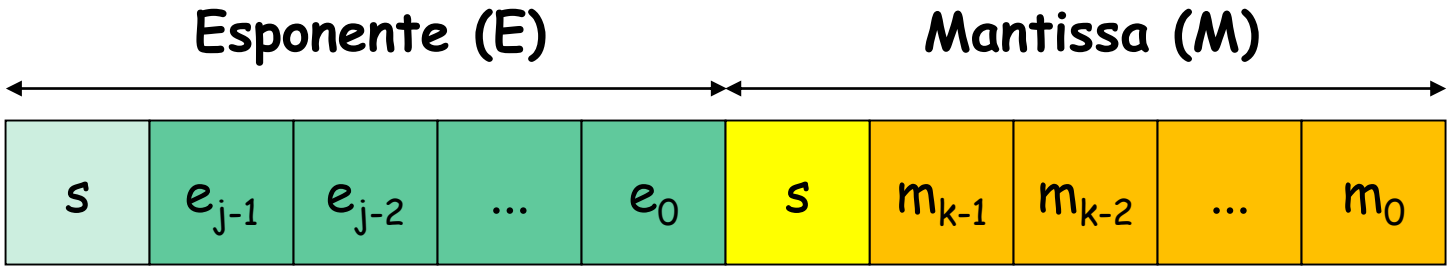
A-B=S, con A, B, S: 4-bit unsigned integers

- A= $(5)_{10}$ = 0101, B= $(2)_{10}$ =0010: ^{2}(-B) =1110 -> S= 0011=$(3)_{10}$, c_{out}=1
- A= $(11)_{10}$ =1011, B= $(9)_{10}$ = 1001: ^{2}(-B) =0111 -> S= 0010= $(2)_{10}$, c_{out}=1
- A= $(2)_{10}$ =0010, B= $(5)_{10}$ =0101: ^{2}(-B) =1011 -> S=1101 =$(13)_{10}$, c_{out}=0
- A= $(9)_{10}$ =1001, B= $(11)_{10}$ =1011: ^{2}(-B) =0101 -> S=1110=$(14)_{10}$, c_{out}=0

- Anche nel caso delle sottrazioni, c_{out} è sufficiente per identificare errori di rappresentazione del risultato (in questo caso: sottraendo > minuendo)
- Al contrario del caso delle addizioni, questi casi sono indicati da $\mathbf{c_{out}=0}$
- L'operazione di sottrazione complementa sempre a 2 il sottraendo (indipendentemente dal caso con/senza segno)

Rappresentazione dei numeri razionali

- Rappresentazione dei numeri in notazione scientifica: tramite esponente (E) e mantissa (M): $N_2 = (M \cdot 2^E)_2$
- Occorre rappresentare ciascuna parte con un certo numero di bit (es. 7+25=32)
- **IEEE 754** è lo standard per la rappresentazione dei numeri in *virgola mobile*



Es.: 32 bit

7-bit (complemento a 2)

25-bit (segno e valore assoluto)

notazione frazionaria normalizzata:
$0{,}5 \leq M < 1$ (tranne lo zero (32 "0")),

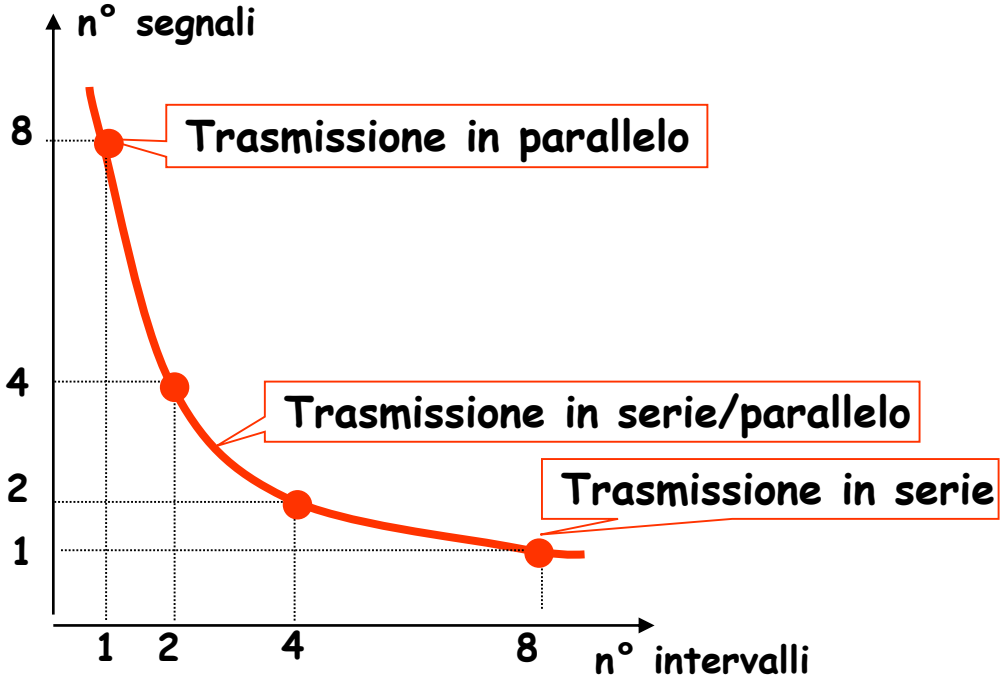
$E_{min} = -2^6 = -64$
$E_{max} = 2^6 - 1 = 63$
$M_{min} = 0{,}5$
$M_{max} = 1 - 2^{-24}$

$$\pm (0{,}5 \cdot 2^{-64} \div (1-2^{-24}) \cdot 2^{63}) \cong \pm (2{,}7 \cdot 10^{-20} \div 0{,}9 \cdot 10^{19})$$

2.4 Trasmissione

Modalità di trasmissione dei bit: compromesso spazio/tempo

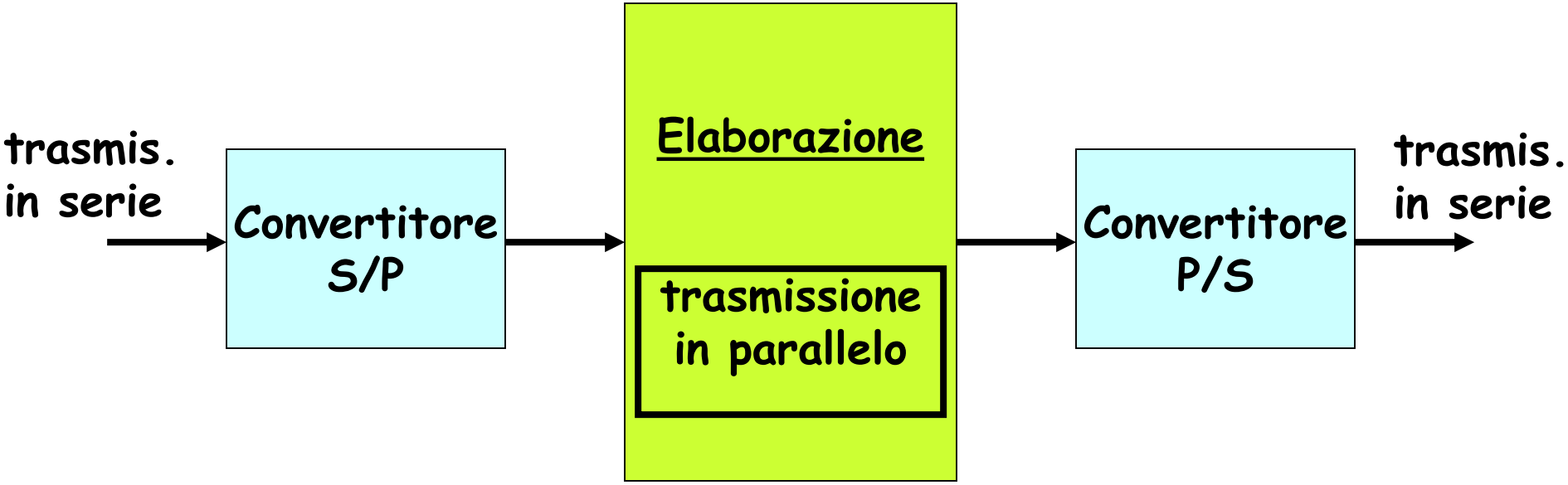
- trasmissione in **parallelo**:
  - n. di segnali = n. bit
  - 1 intervallo di tempo
  - costo più alto, più velocità
- trasmissione in **serie**:
  - 1 segnale binario
  - n intervalli di tempo (tanti quanti i bit)
  - costo più basso, prestazioni più basse
- possibilità di trasmissione serie/parallelo
  - compromesso tra le due modalità



Es.: Codice a 8 bit

Esempio: il primo processore Intel aveva un bus a 4 bit -> per trasmettere dati a multipli di byte era necessario utilizzare la trasmissione in serie/parallelo

Modalità di trasmissione dei bit: convertitori S/P e P/S

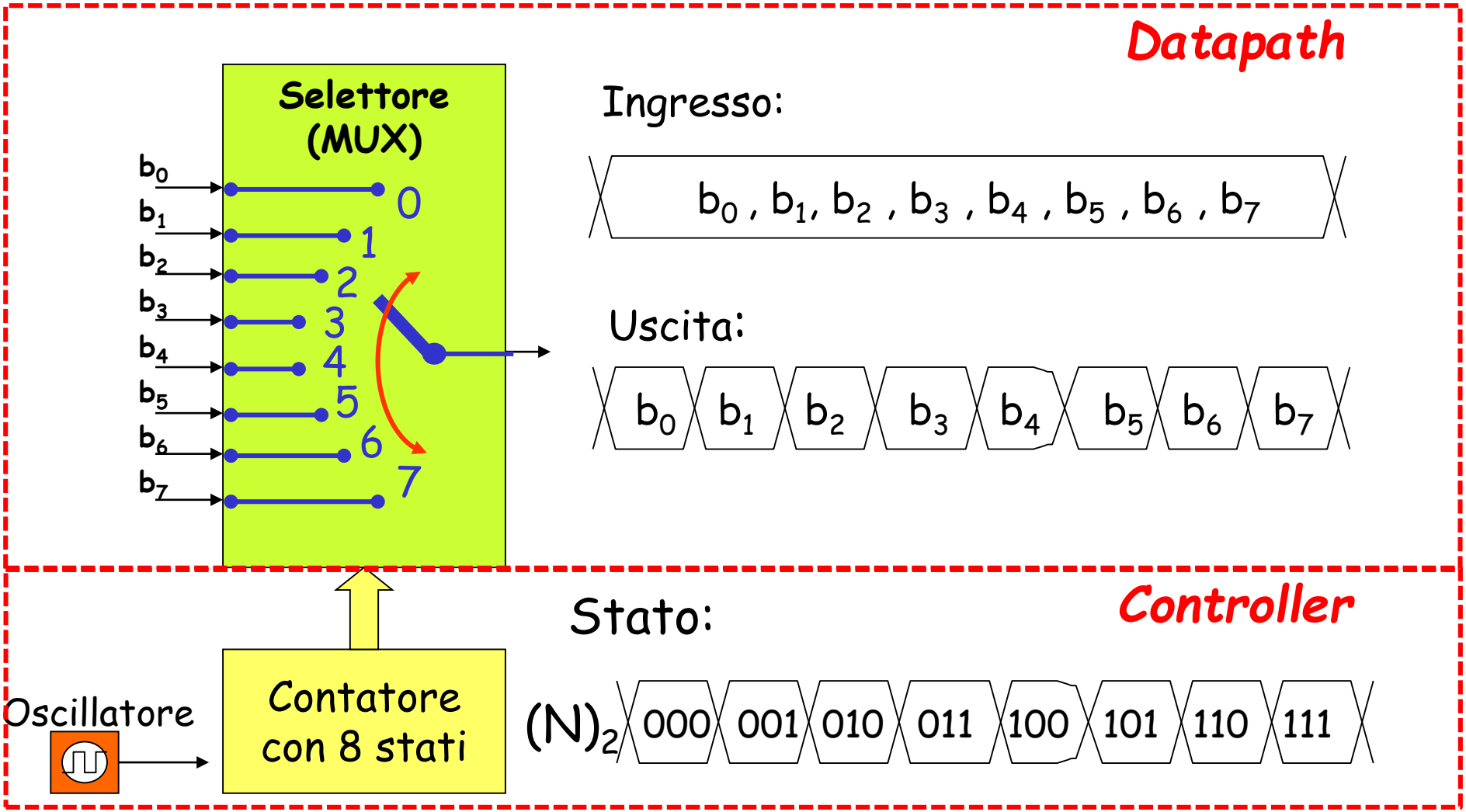


- La modalità di trasmissione all'interno della macchina è di norma in parallelo (per massimizzare la velocità di elaborazione)
- La modalità di trasmissione all'esterno della macchina è di norma in serie (per minimizzare la complessità del supporto fisico)
- I convertitori P/S e S/P sono di norma all'interno del blocco di I/O di una macchina digitale

Esempi: interfaccia di tastiera, interfaccia video

Esempio: conversione P/S di un byte

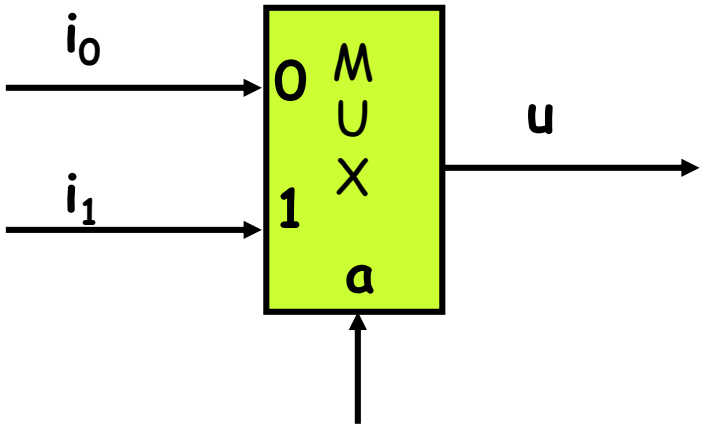
- Il contatore genera in successione i primi 8 numeri binari (da 000 a 111, conteggio modulo 8)

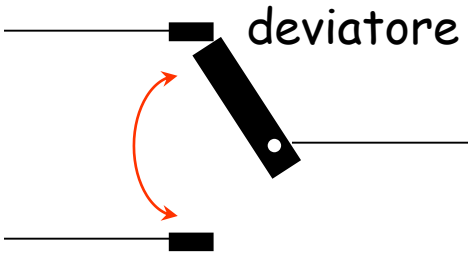


La serializzazione di due bit

- Nella sua accezione più semplice, un **selettore a 2 vie** serializza due soli bit ed ha **1 bit d'indirizzo**
- Il suo comportamento si può descrivere a parole come:

se a=0 allora u=i_0, altrimenti u=i_1



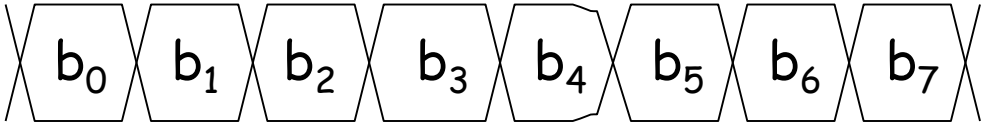


a	i_0	i_1	u
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

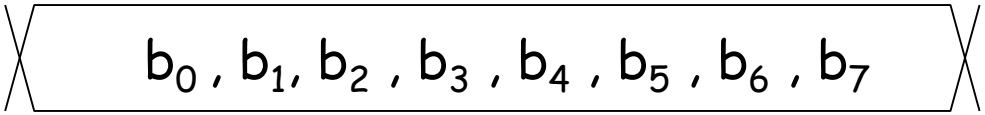
Conversione S/P di un byte

- Duale al MUX, il DEMUX (distributore) realizza la conversione serie/parallelo (S/P)
- Anche in questo caso si utilizza un contatore modulo n (nell'esempio, per un convertitore a 8 bit il contatore è modulo 8)

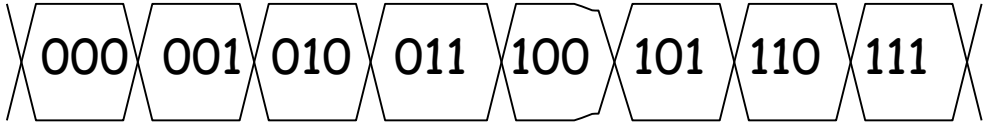
Ingresso:

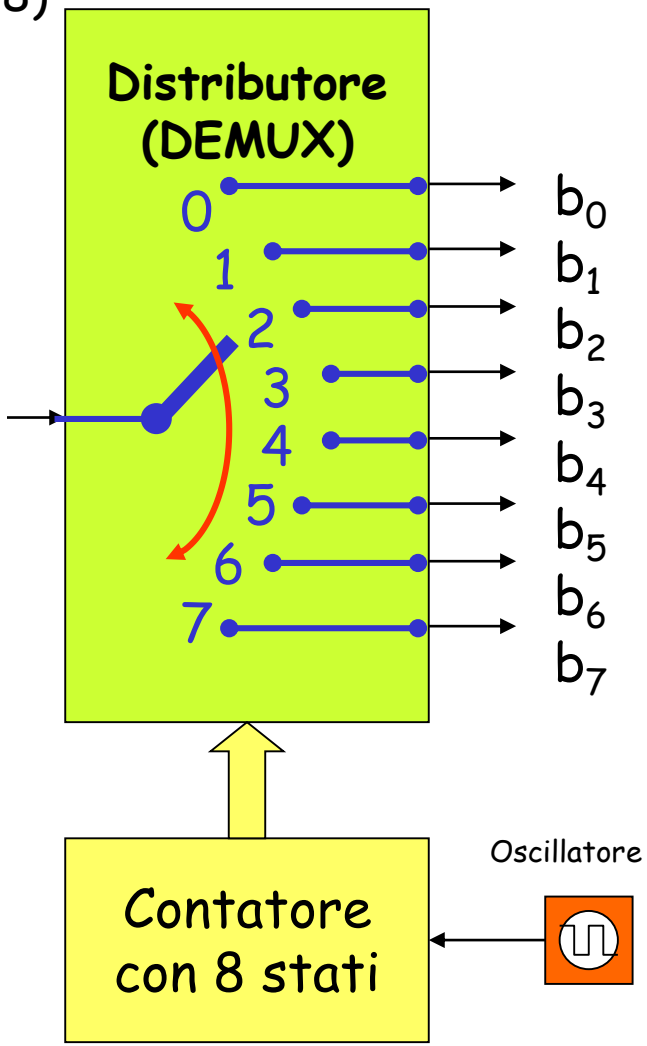


Uscita:



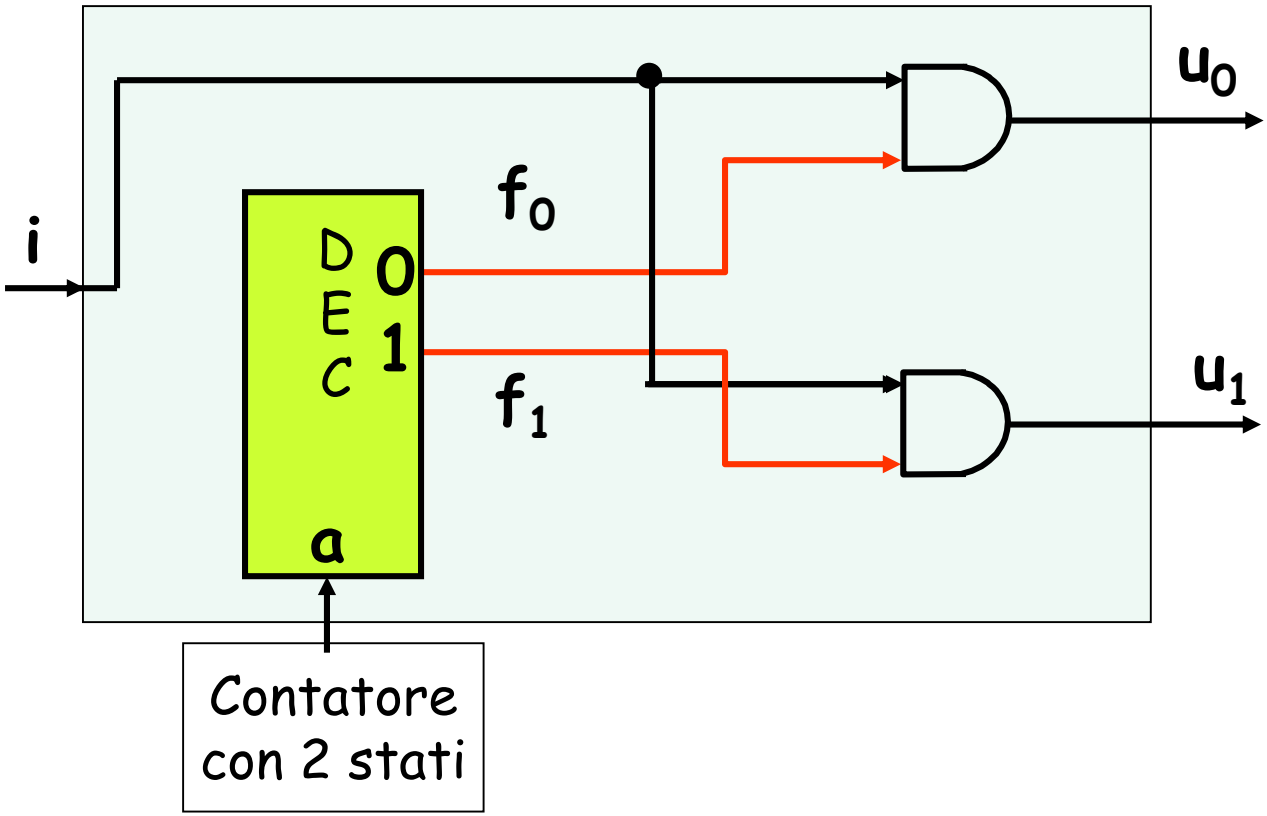
Stato:





La distribuzione di due bit

- Per realizzare un DEMUX si può utilizzare un Decoder
- Il Decoder genera **2** "flag di validità", di cui uno solo alla volta ha valore 1.
- L'uscita che riceve tale valore è la destinazione del bit d'ingresso **i**



a	f_0	f_1
0	1	0
1	0	1

Decoder

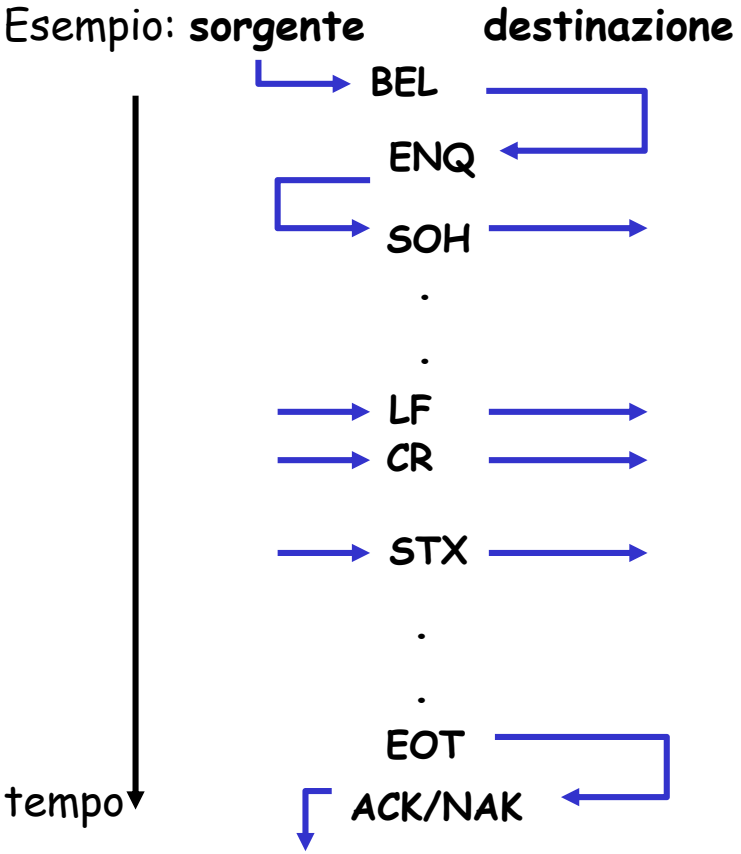
a	u_0	u_1
0	i	0
1	0	i

Demux

Protocolli

- Per aprire, controllare e chiudere un canale di comunicazione digitale, le due macchine coinvolte devono utilizzare un medesimo **protocollo** che permetta loro di scambiarsi prestabiliti **messaggi di servizio**
- Es. Comunicazione tra telescriventi di caratteri ASCII a 7 bit
    - BEL, ENQ, SOH etc. sono messaggi prestabiliti che permettono di procedere nella comunicazione dei caratteri

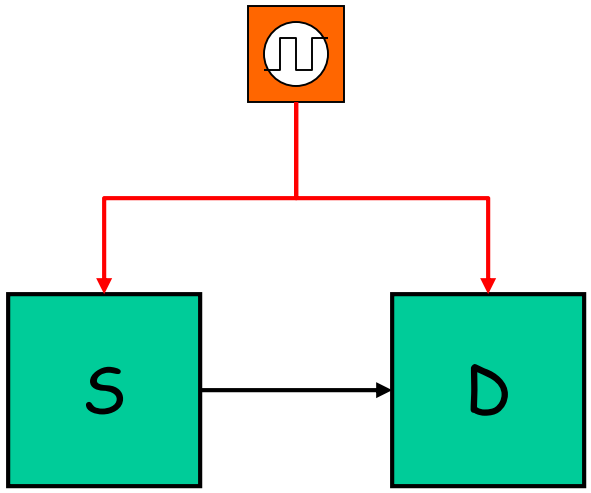
	000	001
0000	NUL	DLE
0001	SOH	DC1
0010	STX	DC2
0011	ETX	DC3
0100	EOT	DC4
0101	ENQ	NAK
0110	ACK	SYN
0111	BEL	ETB
1000	BS	CAN
1001	HT	EM
1010	LF	SUB
1011	VT	ESC
1100	FF	FS
1101	CR	GS
1110	SO	RS
1111	SI	US



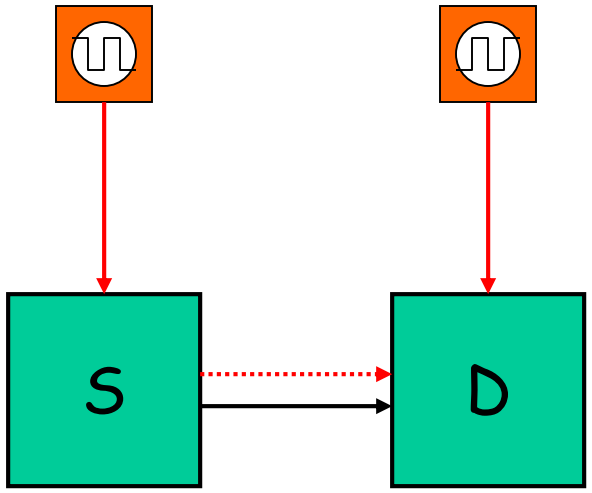
Sincronizzazione

- La comunicazione tra due macchine digitali richiede spesso la condivisione degli istanti di tempo in cui arrivano nuove informazioni (**sincronismo**)
- Il sincronismo può essere ottenuto condividendo lo stesso oscillatore, oppure inviando segnali atti alla sincronizzazione tra le due macchine

"accoppiamento stretto"

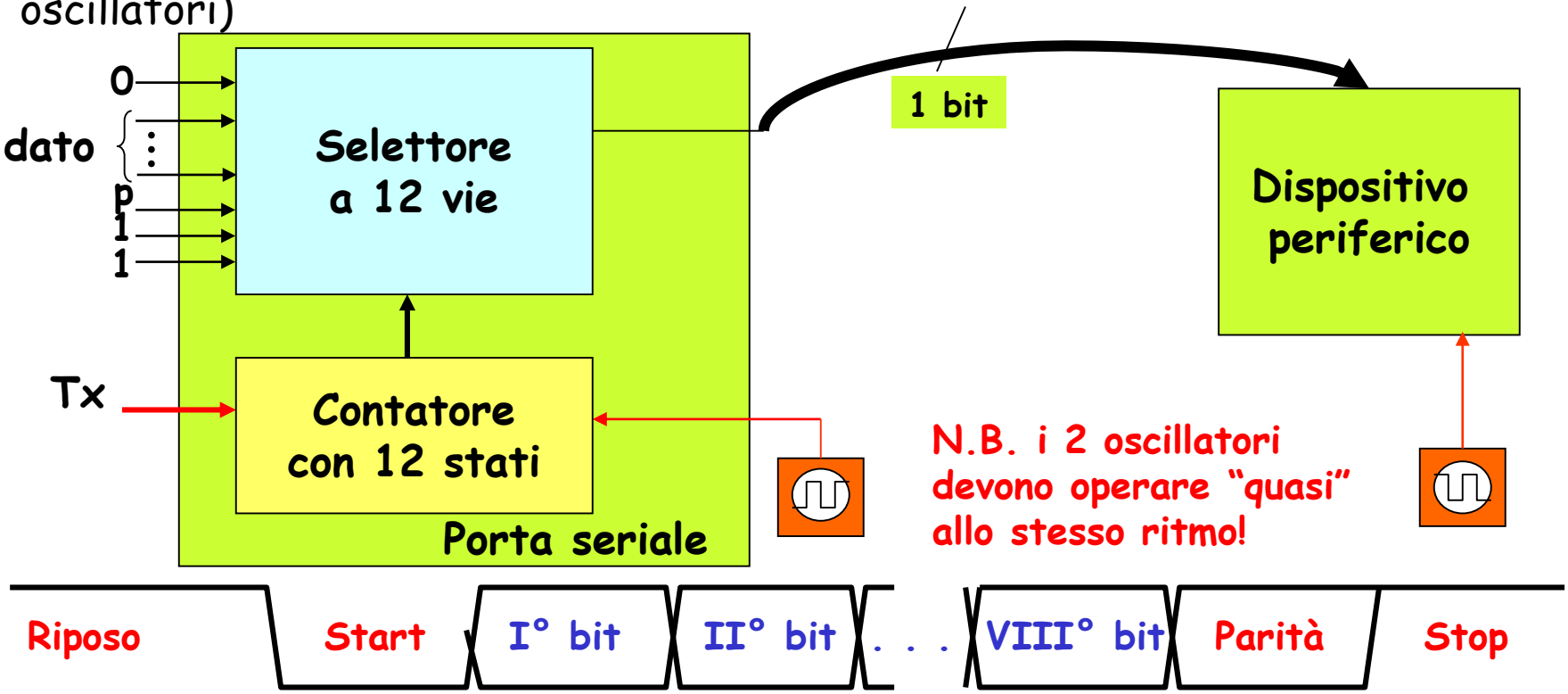


"accoppiamento lasco"



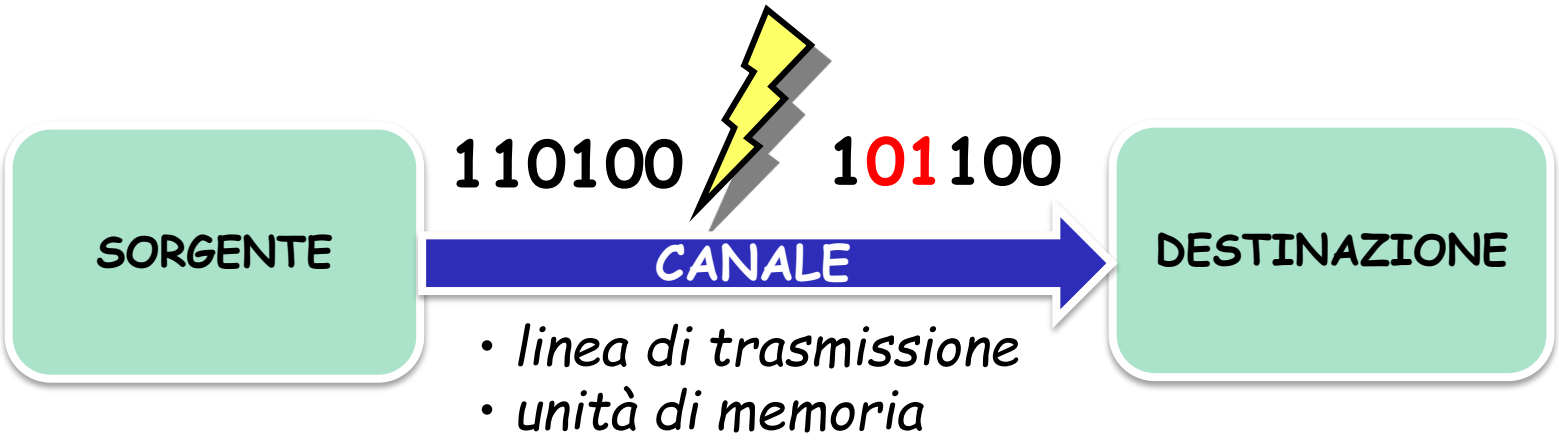
Comunicazione asincrona: il protocollo RS-232

- Protocollo utilizzato per le comunicazioni seriali tra macchine digitali
- Ad es. utilizzato in molte **UART** (Universal Asynchronous Receiver/Transmitter), convertitori S/P e P/S dei microcontrollori
- **Riposo**: valore di tensione alto (in quanto storicamente utilizzato per dimostrare che linea e trasmettitore sono funzionanti)
- **Bit di start**: segnala l'inizio delle comunicazioni (basso)
- Successivamente inviato *in modo seriale* 1 byte + **parità** (v. prossime slides)
- **Due bit di stop**: valore alto, utilizzati per dare tempo al ricevitore di prepararsi alla successiva comunicazione (e «resettare» il ritardo tra i due oscillatori)



2.5 Protezione

Disturbi e Guasti



- I disturbi sul canale di comunicazione modificano in maniera non prevedibile il segnale binario inviato da sorgente a destinazione
- Agendo opportunamente a livello di realizzazione fisica del canale, si può formulare l'ipotesi che l'alterazione di un bit (o errore) nell'ambito di una stringa di **n** bit sia un evento aleatorio
  - a) indipendente dalla posizione del bit nella stringa,
  - b)caratterizzato da una probabilità di occorrenza **p** (tasso di errore).
- Conseguentemente la probabilità che si abbiano **e** errori è data da:

$$P_e = \binom{n}{e} \cdot p^e \cdot (1-p)^{n-e}$$

- Se $n \cdot p << 1 \rightarrow P_0 >> P_1 >> P_2 >>$ … (probabilità sempre più piccola al crescere del numero di errori)

Esempio

Esempio:
p = 1 %
N.B. molto alto!

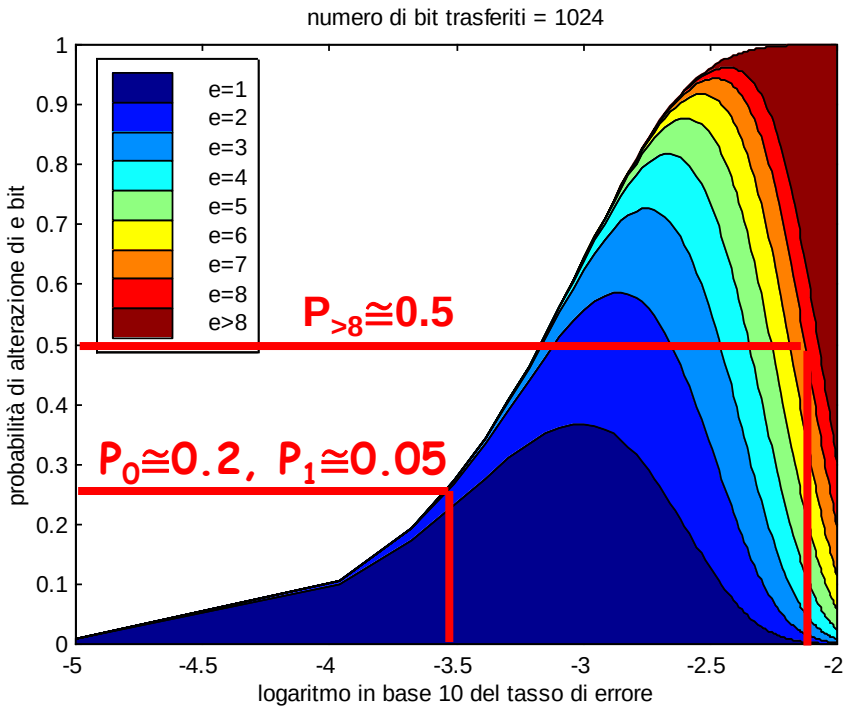
n	P_0	P_1	P_2	P_3
8	*92,27 %*	*7,46 %*	*0,26%*	*0,005 %*
16	*85.14 %*	*13,76 %*	*1,04 %*	*0,049 %*

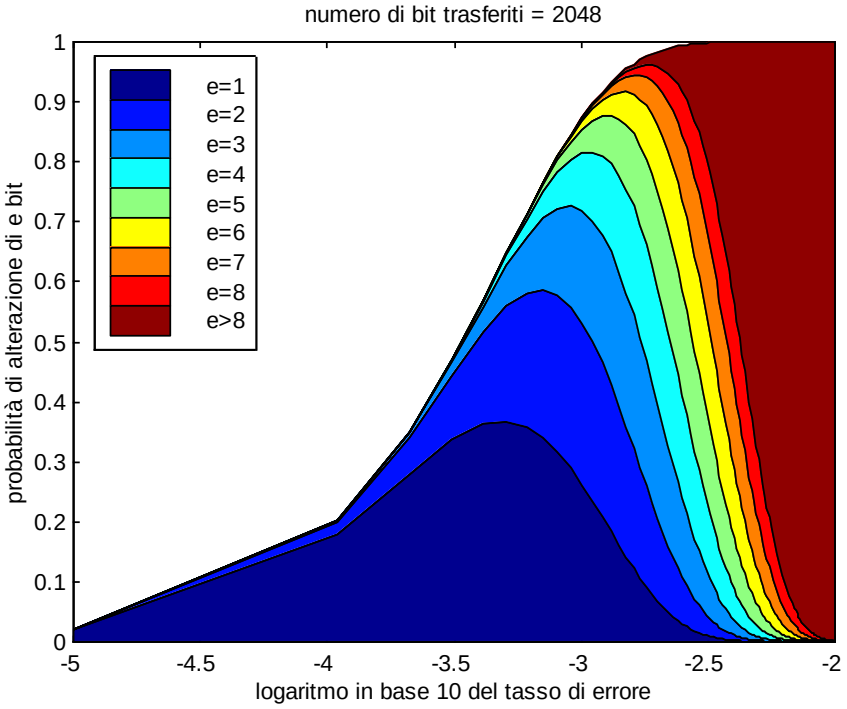
- Per n = 8 le modifiche più probabili riguardano un solo bit (P_0+ P_1=99,99%)
- Per n = 16 le modifiche più probabili riguardano uno o due bit (P_0+ P_1+ P_2= 99,94%)

- Per impedire al ricevitore di prendere decisioni errate, è necessario che **il codice sia ridondante**
- La ricezione di una configurazione «non utilizzata» consente alla destinazione di rilevare un **errore** sul canale

L'ipotesi degli errori indipendenti

- Esempio che mostra P_e con e=1,..,8,>8
  - su sequenze da n=1024 (sx), 2048 (dx) bit
  - al variare di p (in ascissa, da 10^{-5} a 10^{-2})
- All'aumentare di p (da sx. verso dx. in ciascun grafico) aumenta la probabilità di avere un numero maggiore di errori sulla stessa sequenza
- All'aumentare di n (bit trasferiti), a parità di p la probabilità d'errore è maggiore

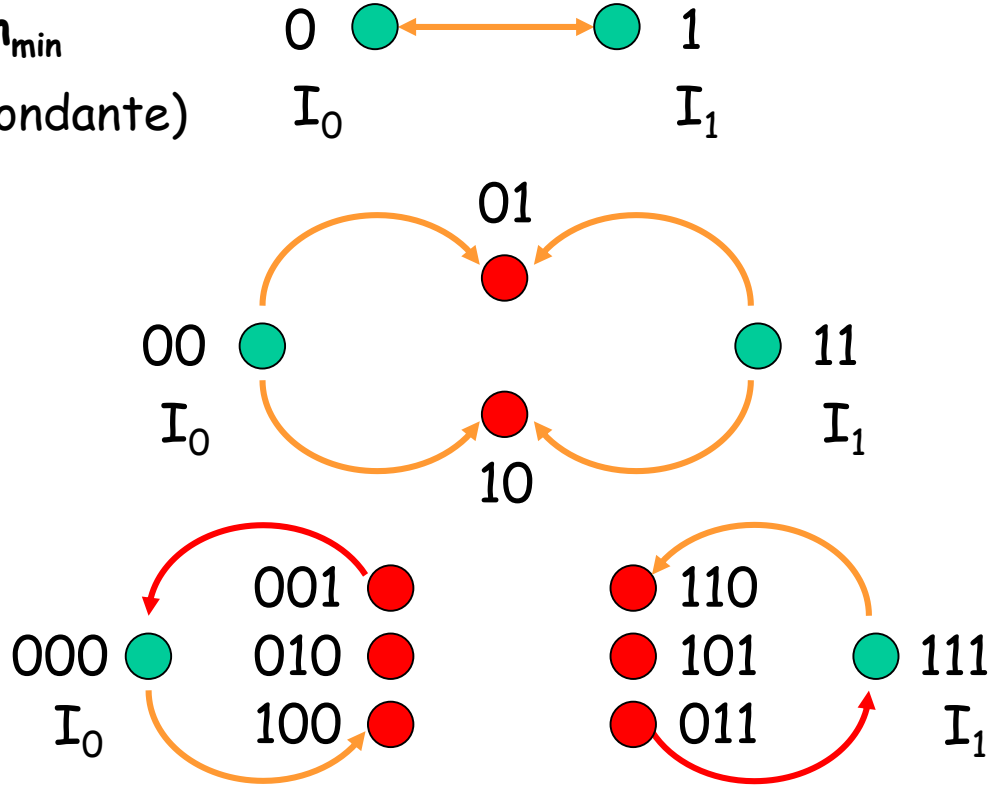




Rilevazione/Correzione di errori singoli (——)

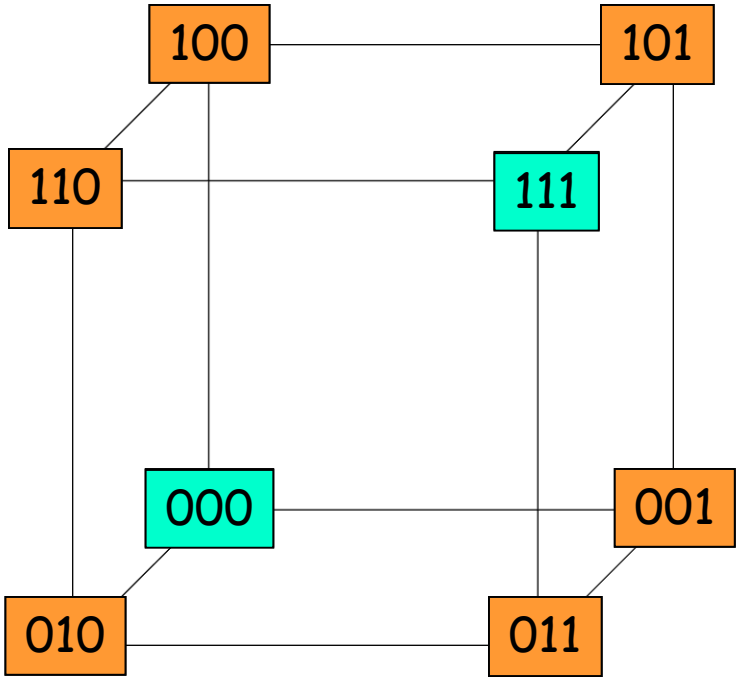
- Alfabeto formato da 2 simboli (M=2)
- Trasmettitore: I_0, ricevitore: I_1

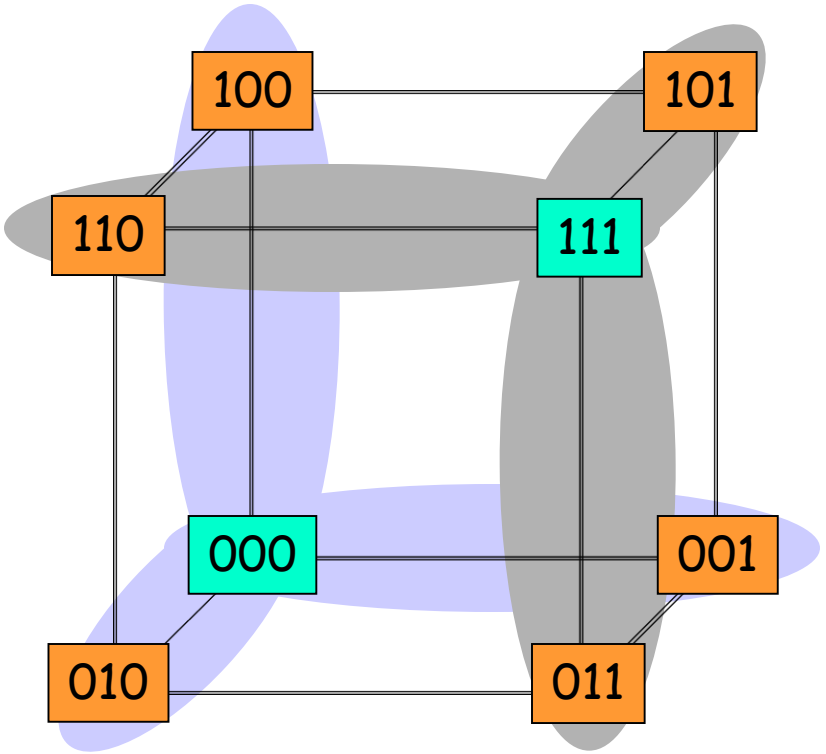
	R	C
$n = 1 = n_{min}$ (codice non ridondante)	NO	NO
$n = 2$	SI	NO
$n = 3$	SI	SI



se $P_1 >> P_2$: correzione errore (——)

Esempio di correzione di errori singoli





Tx trasmette
o NO = 000 o SI = 111

A causa dei disturbi Rx può ricevere una terna qualsiasi

Se $P_1 >> P_2$ ogni configurazione errata può essere corretta

Distanza minima di un codice

Distanza fra due configurazioni binarie A, B di n bit: **D(A,B)**
numero di bit omologhi in A, B con valore diverso
(distanza di Hamming)

Esempi: D(100,101) = 1; D(011,000) = 2; D(010,101) = 3

Distanza minima di un codice C: $D_{MIN}(C)$
valore minimo della distanza tra due qualsiasi configurazioni utilizzate dal codice C.

- **I codici non ridondanti hanno $D_{MIN} = 1$**
- **I codici ridondanti possono avere $D_{MIN} > 1$**

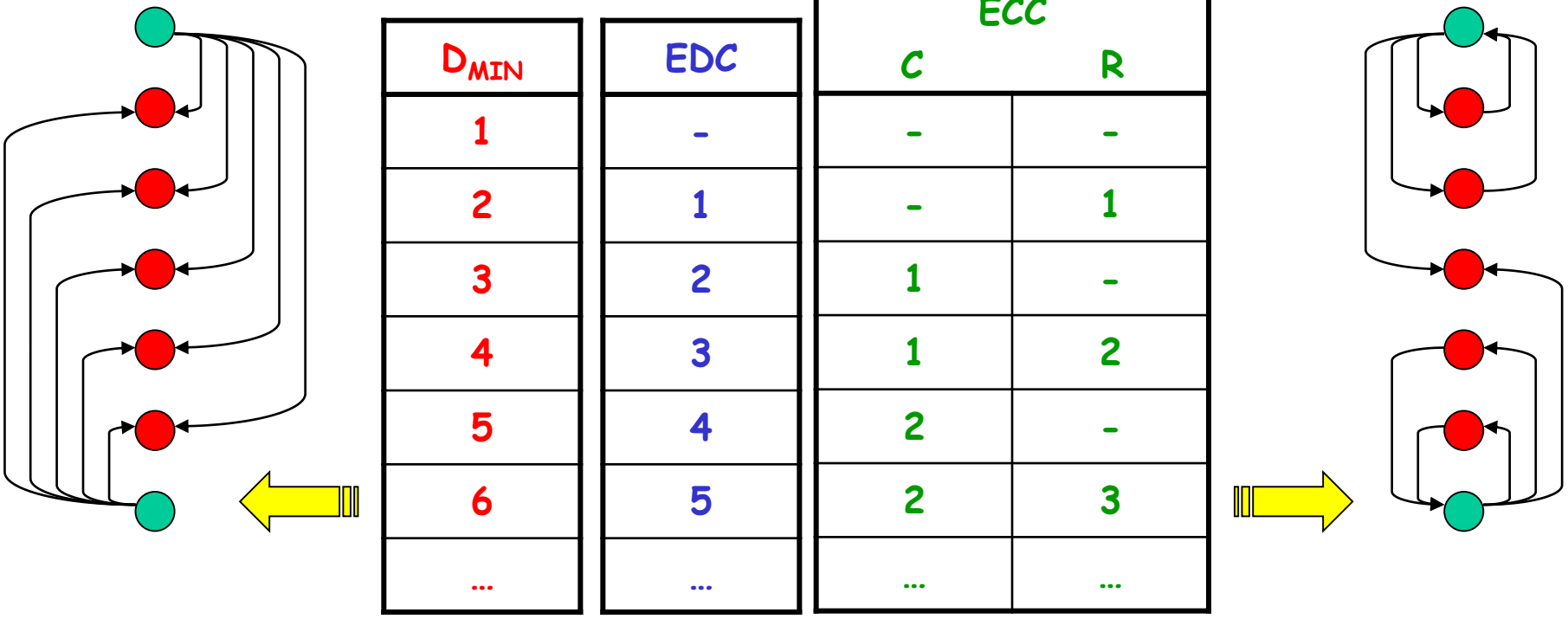
(basta una coppia di configurazioni a distanza 1 perchè $D_{MIN} = 1$)

Esempi:

- D_{MIN} (Codice BCD) = 1
- D_{MIN} (Codice 1/10) = 2
- D_{MIN} (Codice 7-Segmenti) = 1 (si pensi alla coppia «zero»/«otto»)

Error(s) Detecting/Correcting Codes

- Un codice con
  - D_{MIN} = R+1 consente la rilevazione di R errori
  - D_{MIN} = R+C+1 (R≥C) consente la correzione di C errori e la rilevazione di R errori

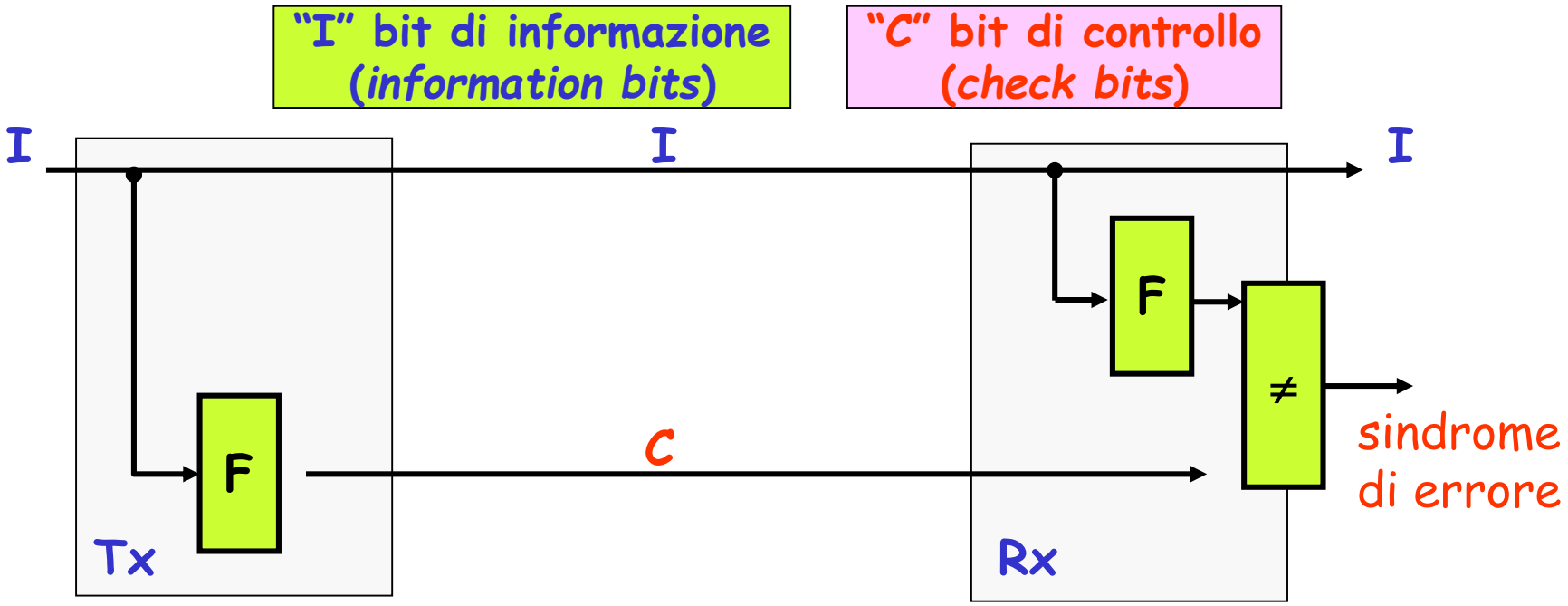


D_{MIN}	EDC	ECC	
		C	R
1	-	-	-
2	1	-	1
3	2	1	-
4	3	1	2
5	4	2	-
6	5	2	3
...	...	...	...

n° max di errori contemporanei: rilevabili | rilevabili e correggibili | rilevabili soltanto

Codici separabili: rilevazione di errori singoli

- **Codice separabile**: ai bit di informazione (I) sono affiancati, in trasmissione, un certo numero di **bit di controllo** (C) calcolati tramite una certa funzione F
- Tali bit di controllo aggiungono ridondanza al codice di partenza aumentando la sua distanza minima
- Il ricevitore calcola F sui bit I ricevuti e confronta il risultato con i C ricevuti generando la sindrome d'errore
- Solo se la sindrome = 0 i bit inviati sono da considerare **integri**



Le due configurazioni 1, 0 della **sindrome di errore** identificano rispettivamente la presenza e l'assenza di un errore singolo.

Codici separabili: correzione di errori singoli

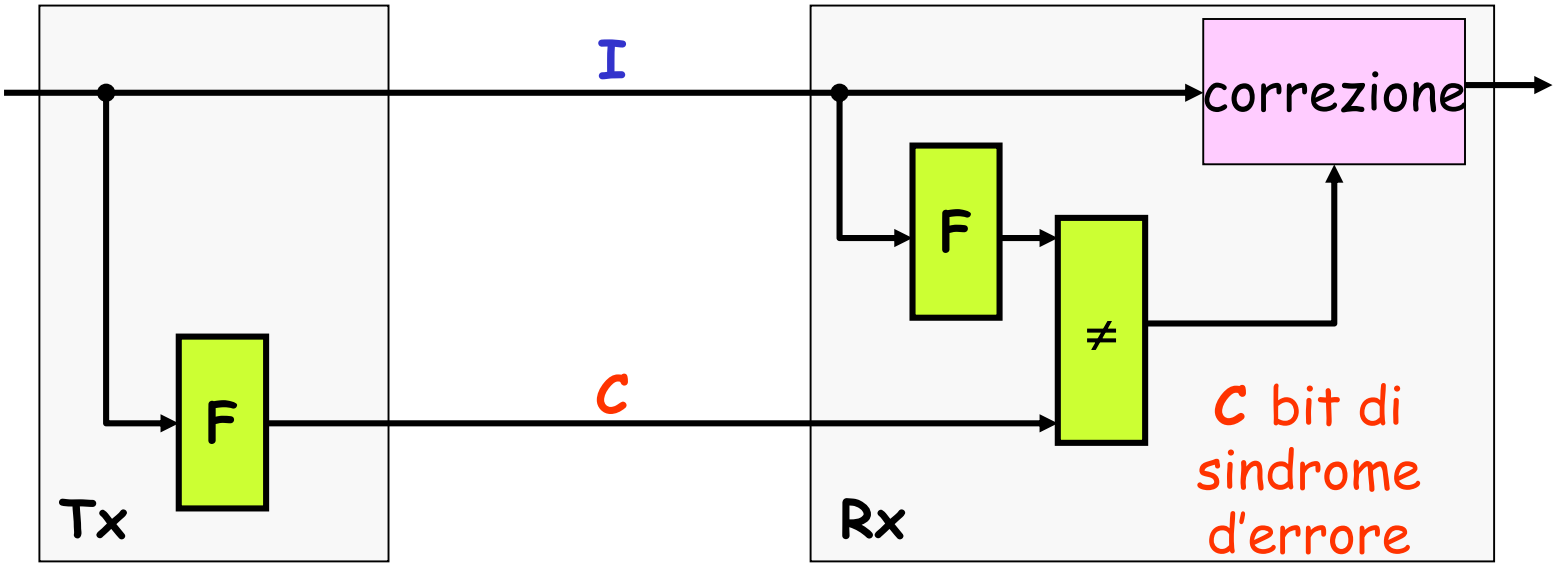
- Per **correggere** gli errori **singoli** oltre a rilevarli, è necessario che:

$$C : 2^C \geq I + C + 1$$

- Es.: con I=1, 1 solo bit di controllo (C=1) non basta in quanto $2^1 < I+C+1 = 3$: occorrono 2 bit di controllo

I	1	2	3	4	5	6	7	…
C	2	3	3	3	4	4	4	…

- A livello del ricevitore si aggiunge un ulteriore blocco che valuta la sindrome d'errore e corregge l'errore



Le 2^C configurazioni delle **sindromi di errore** devono indicare se non c'è errore (1 situazione) e se c'è, dov'è (I + C situazioni).

SINGLE EDC: il bit di parità

Bit di parità p - bit che la sorgente aggiunge ad una stringa di bit di codifica al fine di renderne **pari** il n° di "uni".
Errore di parità e - bit che la destinazione pone a 1 se e solo se riceve una configurazione con un numero **dispari** di "uni".

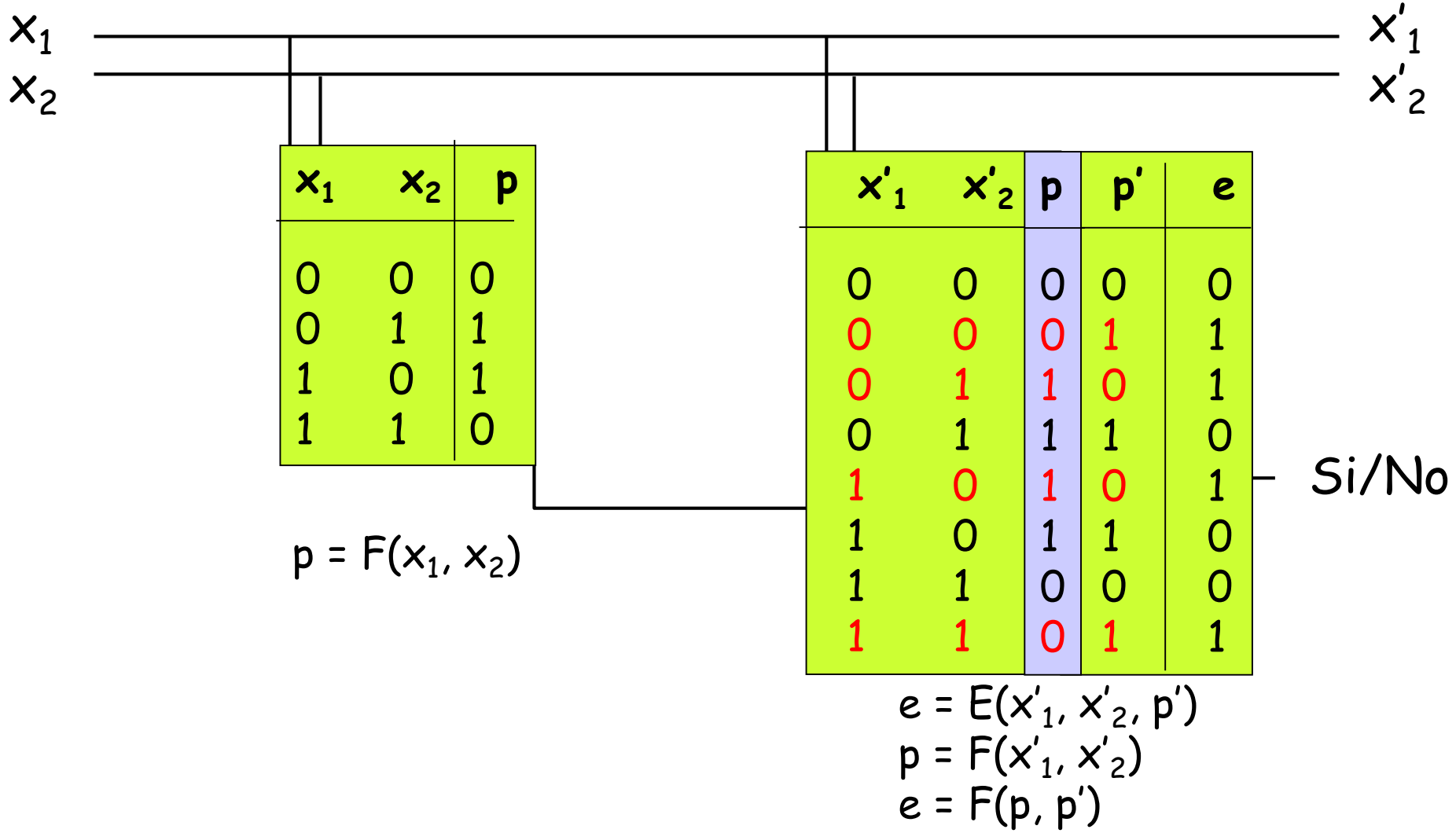
- L'aggiunta del bit di parità permette a un codifica **non ridondante** di avere $D_{MIN} = 2$, dunque di rilevare gli errori singoli
- Rilevati gli errori su 1,3,5 bit (ma non quelli pari!)

x_1	x_2	p
0	0	0
0	1	1
1	0	1
1	1	0

Codice con $D_{MIN} = 2$

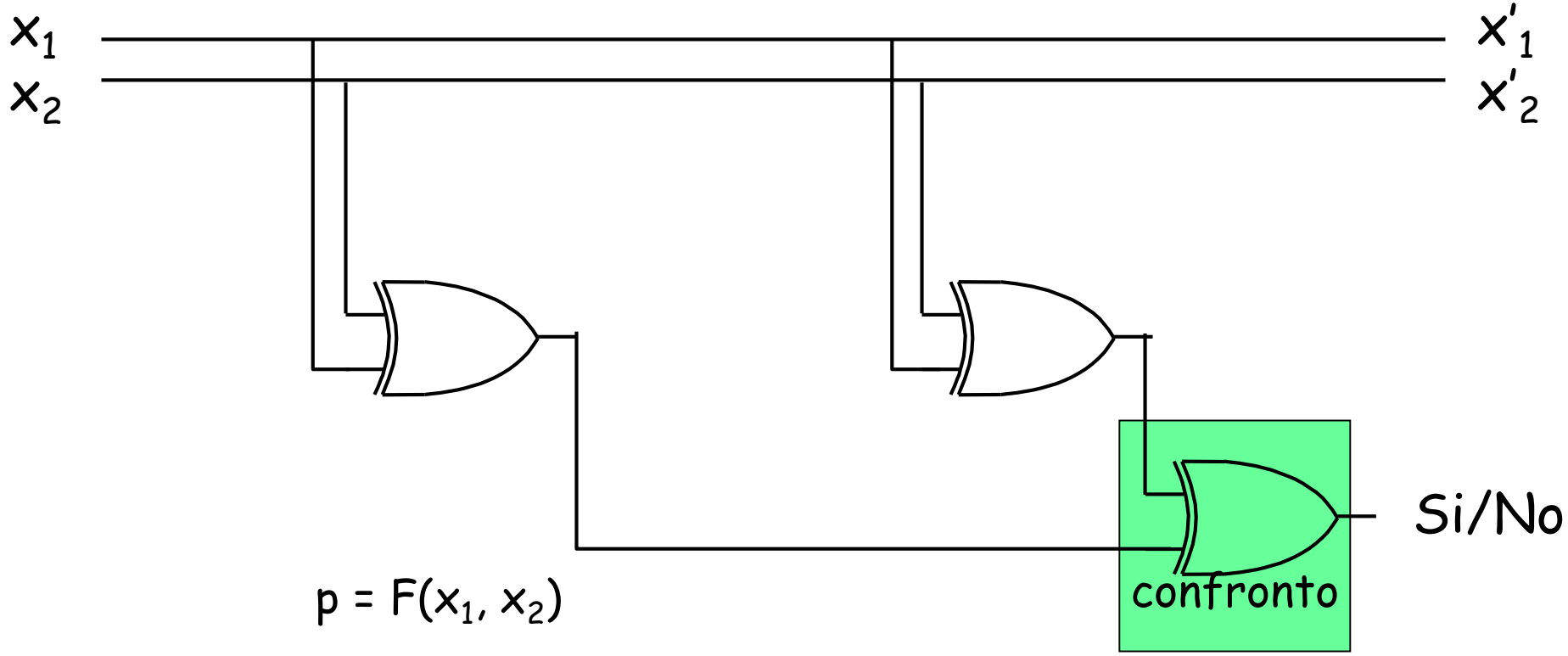
x_1	x_2	p	e
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Calcolo della parità e della sindrome d'errore



- Caso n=2: tramite un ex-or si realizza sia il calcolo del bit di parità (Tx, Rx), sia quello della sindrome d'errore (Rx)

Calcolo della parità e della sindrome d'errore



$e = E(x'_1, x'_2, p')$
$p = F(x'_1, x'_2)$
$e = F(p, p')$

- Caso n=2: tramite un ex-or si realizza sia il calcolo del bit di parità (Tx, Rx), sia quello della sindrome d'errore (Rx)

SINGLE ECC: il codice di Hamming

- Codice di Hamming: calcolo della parità di particolari gruppi di bit
- Rileva e corregge **errori singoli** aggiungendo **3** bit di controllo (H_1 H_2 H_4)

$$I = 4:\ 2^C \geq I + C + 1 \rightarrow C = 3 \rightarrow I + C = 7$$

M=10 info	I=4 (BCD) $X_3X_2X_1X_0$	C=3 $H_4H_2H_1$
0	0000	000
1	0001	111
2	0010	110
3	0011	001
4	0100	101
5	0101	010
6	0110	011
7	0111	100
8	1000	011
9	1001	100

- I 3 bit di controllo permettono di avere $D_{MIN}=3$ e si calcolano come parità sulle seguenti terne:

Tx

$$H_1 = X_3 \oplus X_2 \oplus X_0$$
$$H_2 = X_3 \oplus X_1 \oplus X_0$$
$$H_4 = X_2 \oplus X_1 \oplus X_0$$

- La sindrome di errore si ottiene:

Rx

$$E_1 = X_3' \oplus X_2' \oplus X_0' \oplus H_1'$$
$$E_2 = X_3' \oplus X_1' \oplus X_0' \oplus H_2'$$
$$E_4 = X_2' \oplus X_1' \oplus X_0' \oplus H_4'$$

Se $E_4 = E_2 = E_1 = 0$ **OK**

In caso contrario: **NO**

$E_4\,E_2\,E_1$ = indice del bit affetto da errore e quindi da correggere (complementare);

L'indice è relativo alla seguente disposizione:

1	2	3	4	5	6	7
001	010	011	100	101	110	111
H_1	H_2	X_3	H_4	X_2	X_1	X_0

Esempi:

1	2	3	4	5	6	7
001	010	011	100	101	110	111
H_1	H_2	X_3	H_4	X_2	X_1	X_0

Tx

"9": $X_3X_2X_1X_0 = 1001$

$H_1 = X_3 \oplus X_2 \oplus X_0 = 0$

$H_2 = X_3 \oplus X_1 \oplus X_0 = 0$

$H_4 = X_2 \oplus X_1 \oplus X_0 = 1$

$X_3X_2X_1X_0H_4H_2H_1 = 1001100$

Caso 1: nessun errore

$X_3'X_2'X_1'X_0'H_4'H_2'H_1' = 1001100$

Rx

$E_1 = X_3' \oplus X_2' \oplus X_0' \oplus H_1' = 0$

$E_2 = X_3' \oplus X_1' \oplus X_0' \oplus H_2' = 0$

$E_4 = X_2' \oplus X_1' \oplus X_0' \oplus H_4' = 0$

$E_4\,E_2\,E_1 = 000 \rightarrow X_k = X_k',\ \forall\ k$

$X_3X_2X_1X_0 = 1001$ ("9") **OK**

Caso 2: errore singolo

$X_3'X_2'X_1'X_0'H_4'H_2'H_1' = 0001100$

$E_1 = X_3' \oplus X_2' \oplus X_0' \oplus H_1' = 1$

$E_2 = X_3' \oplus X_1' \oplus X_0' \oplus H_2' = 1$

$E_4 = X_2' \oplus X_1' \oplus X_0' \oplus H_4' = 0$

$E_4\,E_2\,E_1 = 011 \rightarrow X_3 = \text{not } X_3'$

$X_k = X_k',\ \forall\ k \neq 3$

$X_3X_2X_1X_0 = 1001$ ("9") **OK**

Esempi:

1	2	3	4	5	6	7
001	010	011	100	101	110	111
H_1	H_2	X_3	H_4	X_2	X_1	X_0

Tx

"9": $X_3X_2X_1X_0 = 1001$

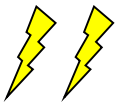
$H_1 = X_3 \oplus X_2 \oplus X_0 = 0$

$H_2 = X_3 \oplus X_1 \oplus X_0 = 0$

$H_4 = X_2 \oplus X_1 \oplus X_0 = 1$

$X_3X_2X_1X_0H_4H_2H_1 = 1001100$

Caso 3: errore doppio



$X_3'X_2'X_1'X_0'H_4'H_2'H_1' = 0011100$

Rx

$E_1 = X_3' \oplus X_2' \oplus X_0' \oplus H_1' = 1$

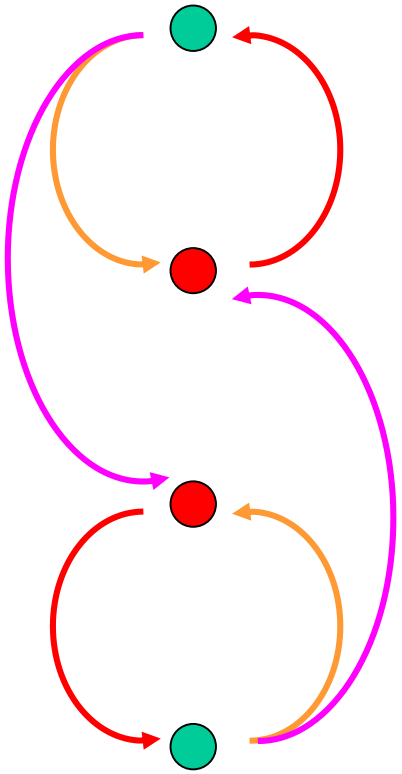
$E_2 = X_3' \oplus X_1' \oplus X_0' \oplus H_2' = 0$

$E_4 = X_2' \oplus X_1' \oplus X_0' \oplus H_4' = 1$

$E_4\,E_2\,E_1 = 101 \rightarrow X_2 = \text{not } X_2'$

$X_k = X_k',\ \forall\ k \neq 2$

$X_3X_2X_1X_0 = 0111$ ("7") **NO**



errore singolo (——)

errore doppio (——)

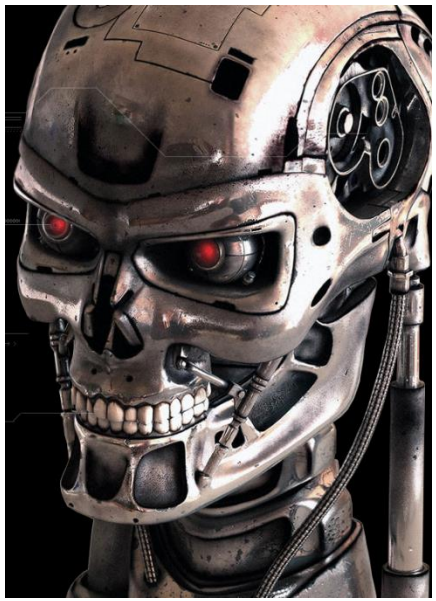
correzione errore (——)

Indovinello

Successivamente alla nascita della singolarità tecnologica, 10 umani vengono catturati da una malvagia intelligenza artificiale. Essa li pone di fronte a un pericoloso gioco

«vi disporrò in fila indiana e metterò poi in testa a ciascuno un cappello, il cui colore sarà casualmente nero o bianco.

Partendo dall'ultimo della fila, ciascuno di voi dovrà dire il colore del cappello che ha in testa. Vi è concesso un solo errore, altrimenti sarete terminati.»



Come fanno gli umani a salvarsi?