UNIVERSITÀ
DEGLI STUDI DI TRIESTE

dipartimento
di ingegneria
e architettura

Corso di Laurea in Tecniche di Radiologia Medica per immagini e Radioterapia

Sistemi Elettronici e informatici in ambito di Imaging I

1CFU – 10 ore

LA CODIFICA DELL'INFORMAZIONE

Prof. Sara Renata Francesca Marceglia

Rappresentazione dell'informazione

Rappresentare le informazioni significa operare

con un insieme limitato di simboli (detto *alfabeto* A)

in modo non ambiguo (algoritmi di traduzione tra codifiche)

Esempio: numeri interi assoluti

Codifica decimale (in base dieci)

Alfabeto (A) = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

"sette" : 7

"ventitre" : 23

"centotrentotto" : 138

La notazione posizionale

Codifica decimale (in base dieci)
Alfabeto (A) = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

"sette" : 7 = 7*1
"ventitre" : 23 = 20 + 3 = 2*10 + 3*1
"centotrentotto" : 138 = 100 + 30 + 8 = 1*100 + 3*10 + 8*1

- ogni cifra corrisponde a una diversa **potenza di dieci**
- **dalla cifra più significativa** a quella meno significativa

$$N = c_{N-1}10^{N-1} + c_{N-2}10^{N-2} + \ldots + c_1 10^1 + c_0 * 10^0$$

B = 10 = base

La notazione posizionale: codifica binaria

Codifica in base B:

$$N = c_{N-1}B^{N-1} + c_{N-2}B^{N-2} + \ldots + c_1B^1 + c_0*B^0$$

$$c_i \in \{0, 1, 2, \ldots, B-1\} \text{ per ogni } 1 \leq i \leq n$$

B = 2

$c_i \in \{ 0, 1 \}$

$$\mathbf{N = c_{N-1}2^{N-1} + c_{N-2}2^{N-2} + \ldots + c_12^1 + c_0*2^0}$$

BIT (“BInary digIT”): unità ELEMENTARE di informazione

Dispositivi che assumono **due** stati

Ad esempio due valori di tensione V_A e V_B

Codifica binaria dell'informazione

- **Quanti oggetti** diversi posso codificare con parole binarie composte da **k bit**?
  - 1 bit: $2^1 = 2$ stati (0, 1) $\Rightarrow$ 2 oggetti
  - 2 bit: $2^2 = 4$ stati (00, 01, 10, 11) $\Rightarrow$ 4 oggetti
  - 3 bit: $2^3 = 8$ stati (000, 001, 010, 011, 100, 101, 110, 111) $\Rightarrow$ 8 oggetti
  - …
  - k bit: 2^k stati $\Rightarrow$ 2^k oggetti

- Se passiamo da una parola binaria di **k bit** ad una parola di **k+1 bit** si raddoppia il numero di oggetti che si possono rappresentare (**2^{k+1}**)

- **Quanti bit** mi servono per codificare N oggetti?
  - $N \leq 2^k \Rightarrow k \geq \log_2 N \Rightarrow$ **$k = \lceil \log_2 N \rceil$**

Codifica binaria dell'informazione

- Ipotesi implicita
  - le parole di un codice hanno tutte la stessa lunghezza.
- Il calcolatore tratta diversi tipi di dati (numeri, caratteri, ecc.) tutti rappresentati con la codifica binaria
- **Problema:** assegnare un codice univoco a tutti gli oggetti compresi in un insieme predefinito.

Esempio

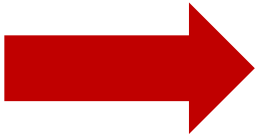
Problema:
assegnare un codice binario univoco a tutti i giorni della settimana

Giorni della settimana: $N = 7 \Rightarrow k \geq \log_2 7 \Rightarrow k = 3$

Con 3 bit possiamo ottenere 8 diverse configurazioni:

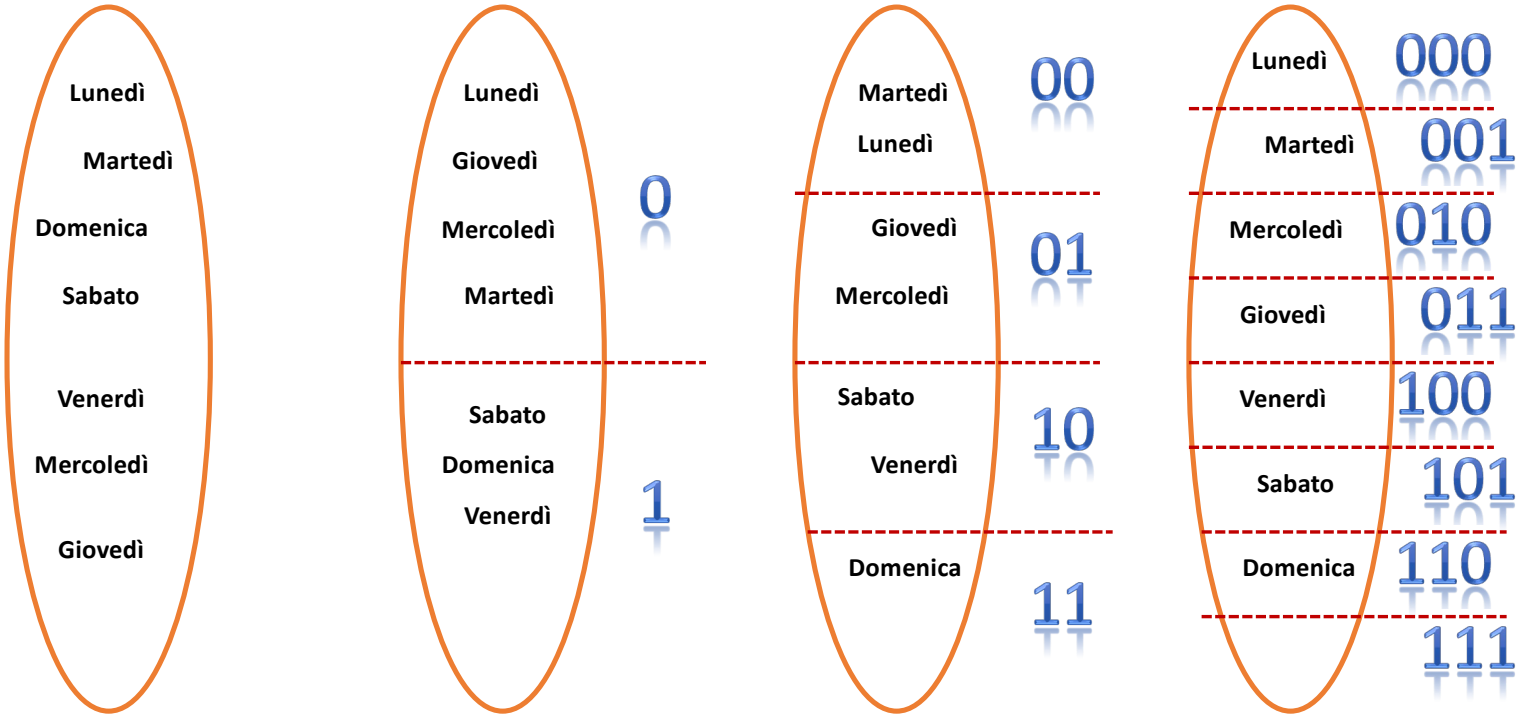
Ne servono 7, quali utilizziamo?

Quale configurazione associamo a quale giorno?



1 - **Identificare** due insiemi:
   Insieme delle **configurazioni ammissibili**;
   Insieme degli **oggetti da rappresentare**.
2 - **Associare** gli elementi dei due insiemi

Esempio: codifica 1

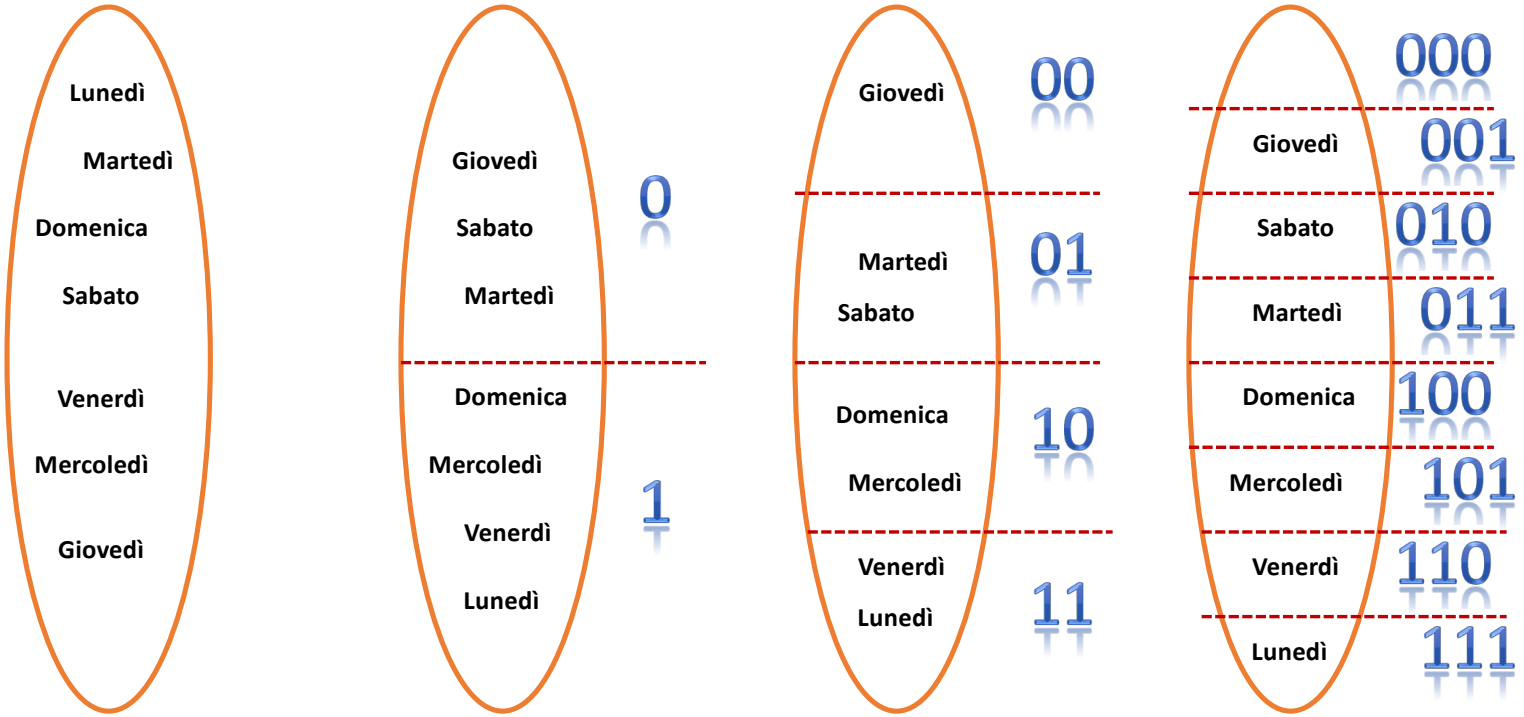


1 bit
2 “gruppi”

2 bit
4 “gruppi”

3 bit
8 “gruppi”

Esempio: codifica 2



Intrinsecamente non univoco:
Deve essere definita
l'assegnazione dei codici

1 bit
2 "gruppi"

2 bit
4 "gruppi"

3 bit
8 "gruppi"

Codifica binaria dei numeri naturali

- Con n bit codifichiamo 2^n numeri: da 0 a 2^n-1
- Con 1 Byte (cioè una sequenza di 8 bit):

  $00000000_{bin} = 0_{dec}$

  $00001000_{bin} = 1 \times 2^3 = 8_{dec}$

  $00101011_{bin} = 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 43_{dec}$

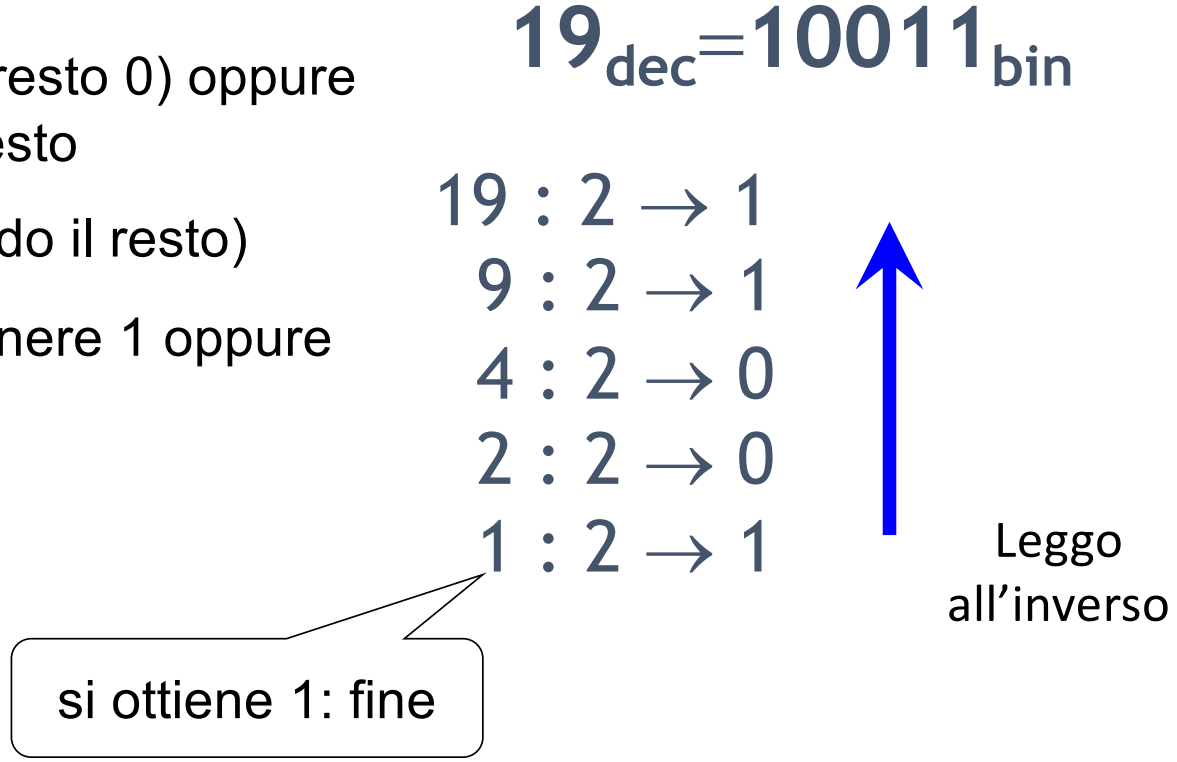
  $11111111_{bin} = \sum_{n=1,2,3,4,5,6,7,8} 1 \times 2^{n-1} = 255_{dec}$
- Conversione bin/dec e dec/bin
  - bin/dec - $\sum_i 2^i$: $11101_{bin} = (2^4+2^3+2^2+2^0) = 29_{dec}$
  - dec/bin - ***metodo dei resti***

Il metodo dei resti

Si calcolano i resti delle divisioni per due

In pratica basta:

1. Decidere se il numero sia pari (resto 0) oppure dispari (resto 1), e annotare il resto
2. Dimezzare il numero (trascurando il resto)
3. Ripartire dal punto 1. fino a ottenere 1 oppure 0 come risultato della divisione
4. Leggere il risultato all'inverso



La notazione posizionale: codifica binaria

29 : 2 = 14 (1)
14 : 2 = 7 (0)
7 : 2 = 3 (1)
3 : 2 = 1 (1)
1 : 2 = **0** (1)

$29_{dec} = 11101_{bin}$

76 : 2 = 38 (0)
38 : 2 = 19 (0)
19 : 2 = 9 (1)
9 : 2 = 4 (1)
4 : 2 = 2 (0)
2 : 2 = 1 (0)
1 : 2 = **0** (1)

$76_{dec} = 1001100_{bin}$

76 si poteva ottenere anche come
76 = 19x4 = **10011**00
Per raddoppiare, in base due, si aggiunge uno zero in coda, così come si fa in base dieci per decuplicare

Casi notevoli

In binario si definisce una *notazione abbreviata*, sulla falsariga del sistema metrico-decimale:

K	$= 2^{10} = 1.024 \approx 10^3$	(Kilo)
M	$= 2^{20} = 1.048.576 \approx 10^6$	(Mega)
G	$= 2^{30} = 1.073.741.824 \approx 10^9$	(Giga)
T	$= 2^{40} = 1.099.511.627.776 \approx 10^{12}$	(Tera)

Esercizi

Che numero intero decimale è rappresentato da
100101
011001
1101010
0110010
1001

Qual è la codifica binaria dei numeri decimali
35
128
67
230
15

Bit e Byte

bit = solo due stati, “0” oppure “1”.

Byte = 8 bit, quindi $2^8 = 256$ stati

KiloByte [KB] = 2^{10} Byte = 1024 Byte ~ 10^3 Byte

MegaByte [MB] = 2^{20} Byte = 1'048'576 Byte ~ 10^6 Byte

GigaByte [GB] = 2^{30} Byte ~ 10^9 Byte

TeraByte [TB] = 2^{40} Byte ~ 10^{12} Byte

PetaByte [PB] = 2^{50} Byte ~ 10^{15} Byte

ExaByte [EB]= 2^{60} Byte ~ 10^{18} Byte

Numeri naturali: codifica con modulo e segno

- Il primo bit a sinistra rappresenta il segno del numero (*bit di segno*), i bit rimanenti rappresentano il valore
  - 0 per il segno positivo, 1 per il segno negativo

- Il bit di segno è *applicato* al numero rappresentato, ma non fa propriamente *parte* del numero
  - il bit di segno non ha significato numerico
  - Inefficiente: due codifiche per lo 0

- Esempi con $n = 9$ (8 bit + un bit per il segno)

  $000000000_{m\&s} = +0 =$

  $000001000_{m\&s} = +1 \times 2^3 = 8_{dec}$

  $100001000_{m\&s} = -1 \times 2^3 = -8_{dec}$

  … e così via …

Numeri interi: complemento a 2

Numeri interi in complemento a 2: il C_2 è *un* sistema binario, ma il primo bit (quello a sinistra, il più significativo) ha *peso negativo*, mentre tutti gli altri bit hanno peso positivo

$$b_n \times (-2^{n-1}) + b_{n-1} \times 2^{n-2} + \ldots + b_1 \times 2^0$$

PESO NEGATIVO

Il bit più a sinistra è ancora chiamato *bit di segno*

Complemento a 2: Esempio

$$000_{C2} = -0\times2^2 + 0\times2^1 + 0\times2^0 = \quad 0_{dec}$$
$$001_{C2} = -0\times2^2 + 0\times2^1 + 1\times2^0 = \quad 1_{dec}$$
$$010_{C2} = -0\times2^2 + 1\times2^1 + 0\times2^0 = \quad 2_{dec}$$
$$011_{C2} = -0\times2^2 + 1\times2^1 + 1\times2^0 = \quad 2+1 = 3_{dec}$$
$$100_{C2} = -1\times2^2 + 0\times2^1 + 0\times2^0 = -4_{dec}$$
$$101_{C2} = -1\times2^2 + 0\times2^1 + 1\times2^0 = -4+1 = -3_{dec}$$
$$110_{C2} = -1\times2^2 + 1\times2^1 + 0\times2^0 = -4+2 = -2_{dec}$$
$$111_{C2} = -1\times2^2 + 1\times2^1 + 1\times2^0 = -4+2+1 = -1_{dec}$$

N.B.: in base al bit di segno lo zero è considerato positivo

Numeri decimali in virgola fissa

- La sequenza di bit rappresentante un **numero frazionario** consta di due parti di lunghezza prefissata
  - I bit a sinistra della virgola rappresentano la parte intera (moltiplicata per 2^E, con E>0)
  - I bit a destra della virgola rappresentano la parte frazionaria (moltiplicata per 2^E, con E<0)
  - Il numero di bit a sinistra e a destra della virgola è stabilito a priori, anche se alcuni bit restassero nulli

- È un sistema di rappresentazione semplice, ma poco flessibile, e può condurre a sprechi di bit

- Per rappresentare in virgola fissa numeri molto grandi (oppure molto precisi) occorrono molti bit

Esempio rappresentazione in virgola fissa

$0{,}1011_{bin}$ (in binario)

$0{,}1011_{bin} = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4} = 1/2 + 1/8 + 1/16 =$

$= 0{,}5 + 0{,}125 + 0{,}0625 = 0{,}6875_{dec}$

$19{,}6875_{dec} = 10011{,}1011$ virgola fissa

poiché si ha:

$19_{dec} = 10011_{bin}$ e $0{,}6875_{dec} = 0{,}1011_{bin}$

proporzione fissa:

5 bit per la parte intera, 4 bit per quella frazionaria

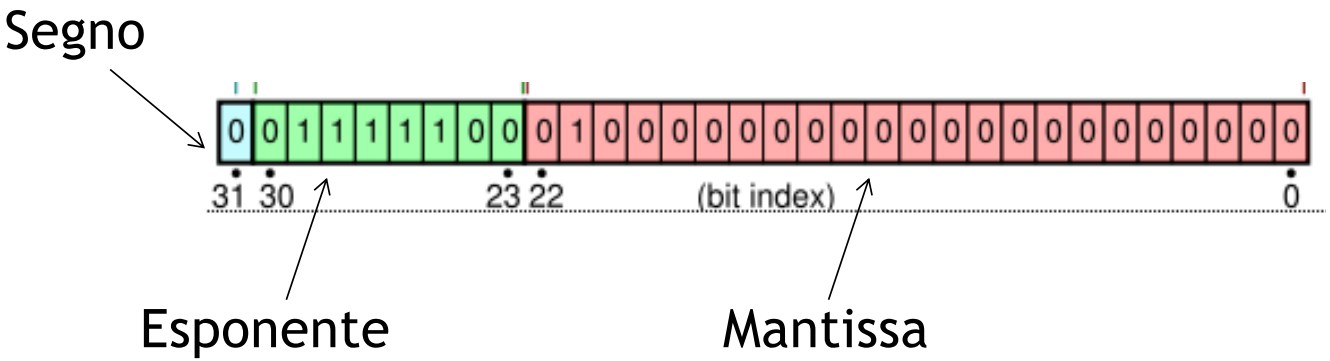
Numeri decimali in virgola mobile

La rappresentazione in *virgola mobile* (o *floating point*) è usata spesso in base 10 (si chiama allora *notazione scientifica*):

$0{,}137 \times 10^8$ notazione scientifica per intendere $13.700.000_{dec}$

La rappresentazione binaria si basa sulla relazione

$$\mathbf{R}_{\text{virgola mobile}} = \mathbf{M} \times \mathbf{2}^{\mathbf{E}}$$

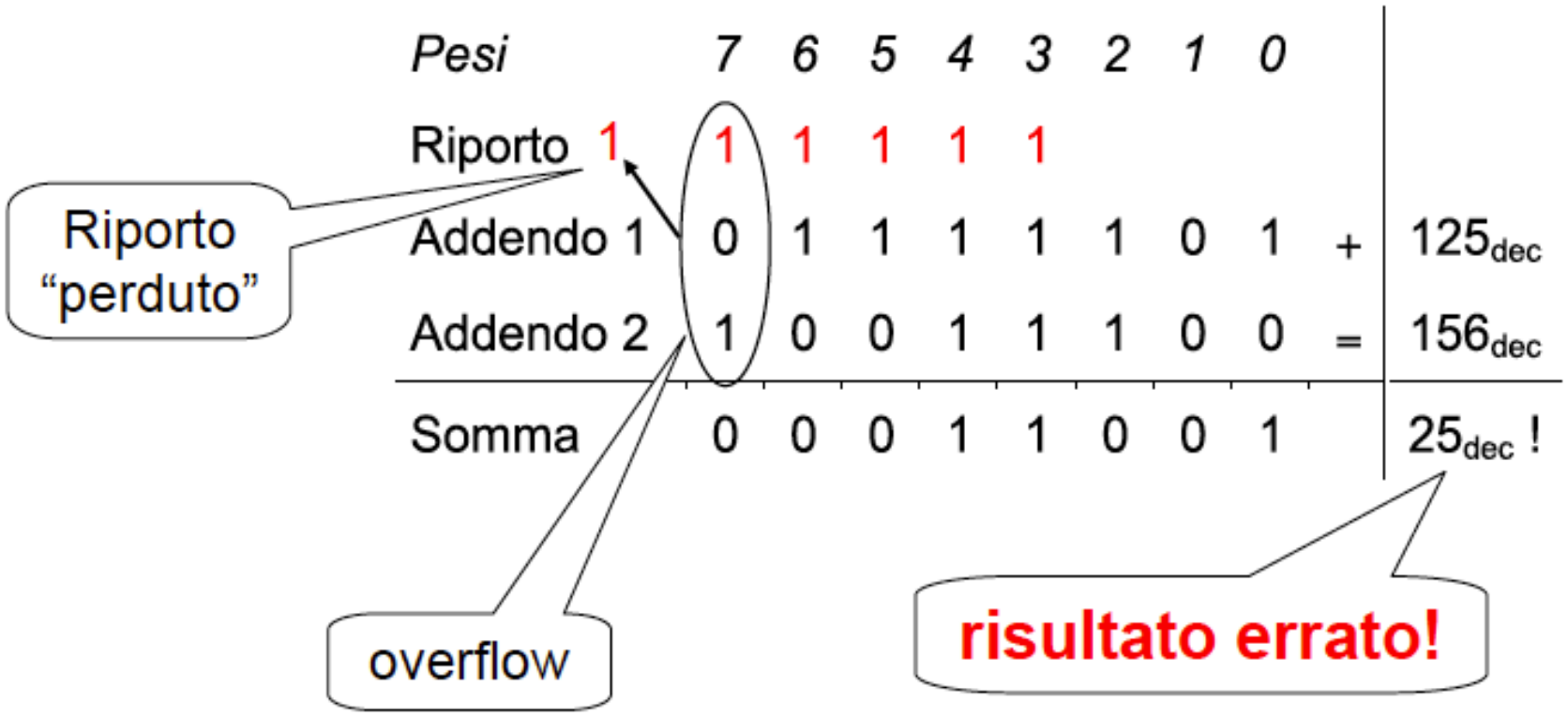


La mantissa e' sempre 1.X, per cui si salva solo X.

Addizione naturale: algoritmo a propagazione dei riporti

Pesi	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*		
Riporto			1	1	1					
Addendo 1	0	1	0	0	1	1	0	1	+	77_{dec}
Addendo 2	1	0	0	1	1	1	0	0	=	156_{dec}
Somma	1	1	1	0	1	0	0	1		233_{dec}

Overflow



addizione **naturale** con overflow

Codifica esadecimale

Codifica in base B:

$$N = c_{N-1}B^{N-1} + c_{N-2}B^{N-2} + \ldots + c_1B^1 + c_0*B^0$$

$$c_i \in \{0, 1, 2, \ldots, B-1\} \text{ per ogni } 1 \leq i \leq n$$

$B = 16$

$c_i \in \{ 0, 1,2,3,4,5,6,7,8,9,A,B,C,D,E,F \}$

$$\mathbf{N = c_{N-1}16^{N-1} + c_{N-2}16^{N-2} + \ldots + c_1 16^1 + c_0*16^0}$$

ATTENZIONE!

IL NUMERO 312 IN BASE ESADECIMALE è

$(312)_{16} = 3*16^2+1*16^1+ 2*16^0= 786$

Codifica esadecimale: Il metodo dei resti

Si calcolano i resti delle divisioni per 16

In pratica basta:

1. Dividere il numero decimale per 16 e annotare quoziente e resto
2. Dividere il quoziente per 16 e annotare il resto
3. Ripartire dal punto 1 fino a ottenere 0 come risultato della divisione
4. Sostituire i resti da 10 a 15 con le lettere A-F
5. Leggere il risultato all'inverso

$(7335)_{10}=???_{16}$

7335:16 = 458 → 7
458 : 16= 28 → 10 **A**
28 : 16 = 1 → 12 **C**
1 : 16 = 0 → 1

si ottiene 0: fine

Leggo all'inverso: **1CA7**

ESERCIZI

BASE 16

$(164)_{16}$ →

164 →

$(A17C)_{16}$ →

325 →

$(1024)_{16}$ →

1024 →

BASE 2

1011

11010010

21

173

Codifica binaria dei caratteri

- Quanti sono gli oggetti da rappresentare?
  - 26 lettere maiuscole
  - 26 lettere minuscole
  - 10 cifre
  - Circa 30 simboli d'interpunzione (, ; …)
  - Circa 30 caratteri di controllo (EOF, CR, …)

- Totale circa 120 oggetti complessivi

  $\Rightarrow$ $\mathbf{k = \lceil \log_2 120 \rceil = 7.}$

Codifica binaria dei caratteri

- **Codice ASCII** (American Standard Code for Information Interchange) utilizza **7 bit**
  $\Rightarrow$ può rappresentare $2^7 = 128$ caratteri detti caratteri ASCII Standard.
- **Codice ASCII esteso** utilizza **8 bit (1 Byte)**
  $\Rightarrow$ può rappresentare $2^8 = 256$ caratteri detti caratteri ASCII estesi.
  - Tale codice comprende i caratteri ASCII standard e alcuni caratteri semigrafici (cornici, lettere nazionali, simboli matematici, ecc.)
- **Codice UNICODE** utilizza **16 bit (2 Byte)**
  - Utile nel caso di alfabeti particolarmente complessi quale quello cinese

Codifica binaria dei caratteri

Codice ASCII standard 7 bit

Byte	Cod.	Char
00000000	0	Null
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

Esempio

Un testo può essere convertito in una successione di numeri grazie al Codice ASCII:
spazio → 32; 'A' → 65; 'B' → 66; …; 'a' → 97; 'b' → 98; …

e quindi: "ciao mondo" → 99 105 97 111 32 109 111 110 100 111

Data questa codifica, un problema come:
trasformare una frase scrivendo con l'iniziale maiuscola tutte le parole che la compongono
(per cui "ciao mondo" dovrebbe diventare "Ciao Mondo")
è effettivamente un problema di calcolo → equivale a sostituire le lettere «c» e «m» in «C» e «M»