



# Rappresentazione delle informazioni

Fondamenti di Informatica, AA 2022/23

Luca Cassano

[luca.cassano@polimi.it](mailto:luca.cassano@polimi.it)



# Cos'è l'informatica?

*Scienza della rappresentazione e dell'elaborazione dell'informazione.*



# Cos'è l'Informatica?

*Scienza della rappresentazione e dell'elaborazione dell'informazione.*

- **Scienza:** ovvero una **conoscenza sistematica e rigorosa** di tecniche e metodi.
- **Informazione:** l'oggetto dell'investigazione scientifica (informazione intesa come entità astratta e come tecnologie per la sua gestione)
- **Rappresentazione:** il modo in cui l'informazione viene strutturata e trasformata in dati fruibili da macchine
- **Elaborazione:** uso e trasformazione dell'informazione per un dato scopo. L'elaborazione deve poter essere eseguita da macchine che processano dati.

[da «Informatica Arte e Mestiere»]



# Rappresentazione dell'informazione



## Il bit

Il *bit* è la più piccola quantità di informazione rappresentabile



## Il bit

Il *bit* è la più piccola quantità di informazione rappresentabile  
Quante configurazioni di  $m$  bit esistono?



## Il bit

Il *bit* è la più piccola quantità di informazione rappresentabile

Quante configurazioni di  $m$  bit esistono?

Quante combinazioni di  $m$  elementi posso realizzare se ogni elemento lo scelgo tra 2 elementi diversi?  $\rightarrow 2^m$

Il *bit* è la più piccola quantità di informazione rappresentabile

Qual è il numero massimo che posso scrivere con  $m$  bit?

Quante combinazioni di  $m$  elementi posso realizzare se ogni elemento lo scelgo tra 2 elementi diversi?  $\rightarrow 2^m$

Ad esempio:

Con 1 cifra in base 2, copro  $[0, 2^1 - 1]$  (cioè  $[0, 1]$ )

Con 4 cifre in base 2, copro  $[0, 2^4 - 1]$  (cioè  $[0, 15]$ )

Con 8 cifre in base 2, copro  $[0, 2^8 - 1]$  (cioè  $[0, 255]$ )

Con 16 cifre in base 2, copro  $[0, 2^{16} - 1]$  (cioè  $[0, 65535]$ )

In binario ho bisogno di molte più cifre rispetto al decimale per esprimere gli stessi numeri





## Il bit

$2^0$	$2^1$	$2^2$	$2^3$	$2^4$	$2^5$	$2^6$	$2^7$	$2^8$	$2^9$	$2^{10}$
1	2	4	8	16	32	64	128	256	512	1024

- Byte = 8 bit =  $2^3$  bit
- KiloByte (kB) =  $10^3$  Byte =  $2^{13}$  bit
- MegaByte (MB) =  $10^6$  Byte =  $2^{23}$  bit
- GigaByte (GB) =  $10^9$  Byte =  $2^{33}$  bit
- TheraByte (TB) =  $10^{12}$  Byte =  $2^{43}$  bit



## Codifica dei numeri

Utilizziamo la definizione di numero in notazione posizionale

$$(N)_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

Es.

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$$

$$\begin{aligned} (1100010)_2 &= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2 = \\ &= 64 + 32 + 2 = (98)_{10} \end{aligned}$$

Nel corso vedremo diverse codifiche per

- I numeri naturali
- I numeri interi (positivi e negativi)
- I numeri reali



## Rappresentazione dei Caratteri

Ogni carattere viene mappato in un numero intero (che è espresso da sequenza di bit) utilizzando dei codici

Il codice più usato è l'*ASCII* (*American Standard Code for Information Interchange*) a 8 bit che contiene:

- Caratteri alfanumerici
- Caratteri simbolici (es. punteggiatura, @&%\$ etc..)
- Caratteri di comando (es. termina riga, vai a capo, tab)



## La codifica ASCII (esempi)

A  $\Leftrightarrow$  65  $\Leftrightarrow$  01000001

a  $\Leftrightarrow$  97  $\Leftrightarrow$  01100001

.

.

.

Z  $\Leftrightarrow$  90  $\Leftrightarrow$  01011010

z  $\Leftrightarrow$  122  $\Leftrightarrow$  01111010



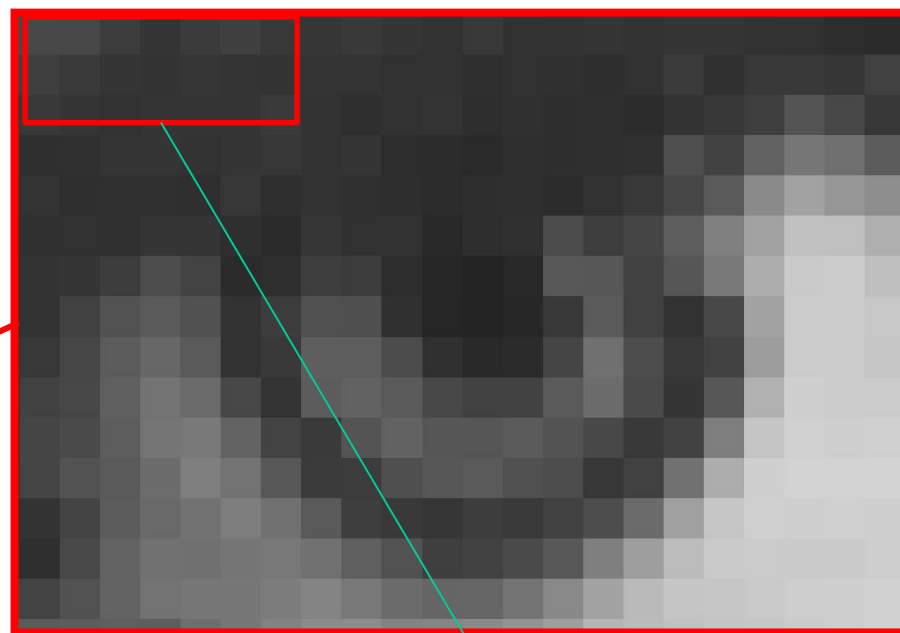
## La codifica ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z



# Codifica delle Immagini

- Le immagini nei calcolatori sono digitali, i.e. tabella di pixel, ciascuno caratterizzato da uno o più valori di intensità.



123	122	134	121	132	133	145	134
122	121	125	132	124	121	116	126
119	127	137	119	139	127	128	131



# Codifica delle Immagini

- Le immagini nei calcolatori sono digitali, i.e. tabella di pixel, ciascuno caratterizzato da uno o più valori di intensità.



Canale rosso



Canale verde



Canale blu



# Rappresentazione dei numeri naturali





## Codifica dei Numeri in Base 10

- Le cifre che abbiamo a disposizione sono 10

$$A_{10} = \{0, 1, \dots, 9\}$$

- Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente



## Codifica dei Numeri in Base 10

- Le cifre che abbiamo a disposizione sono 10
$$A_{10} = \{0, 1, \dots, 9\}$$
- Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente
- Es numero di 4 cifre
  - $3401 = 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$



## Codifica dei Numeri in Base 10

- Le cifre che abbiamo a disposizione sono 10
$$A_{10} = \{0, 1, \dots, 9\}$$
- Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente
- Es numero di 4 cifre
  - $3401 = 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$
  - $4310 = 4 \times 10^3 + 3 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$



## Codifica dei Numeri in Base 10

- Le cifre che abbiamo a disposizione sono 10

$$A_{10} = \{0, 1, \dots, 9\}$$

- Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente
- Es numero di 4 cifre
  - **3401** =  $3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$
  - **4310** =  $4 \times 10^3 + 3 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$
  - **0413** =  $0 \times 10^3 + 4 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$



## Codifica dei Numeri in Base 10

- Le cifre che abbiamo a disposizione sono 10
$$A_{10} = \{0, 1, \dots, 9\}$$
- Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente
- Es numero di 4 cifre
  - $3401 = 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$
  - $4310 = 4 \times 10^3 + 3 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$
  - $0413 = 0 \times 10^3 + 4 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$
- Con  $m$  cifre posso rappresentare  $10^m$  numeri distinti:
$$0, \dots, 10^m - 1$$



# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$



# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$
  - In base 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$



# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$
  - In base 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$
  - In base 8, il dizionario è  $A_8 = \{0, \dots, 7\}$





# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$
  - In base 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$
  - In base 8, il dizionario è  $A_8 = \{0, \dots, 7\}$
  - In base 2, il dizionario è  $A_2 = \{0, 1\}$



# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$
  - In base 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$
  - In base 8, il dizionario è  $A_8 = \{0, \dots, 7\}$
  - In base 2, il dizionario è  $A_2 = \{0, 1\}$
- Un numero è una sequenza di cifre
$$a_n a_{n-1} \dots a_1 a_0 \text{ con } a_i \in A$$
  - 8522 è una sequenza di 4 cifre di  $A_{10}$ ,  $\{8, 5, 2\} \subset A_{10}$ .



# Codifica dei Numeri in una Base Qualsiasi

- Ogni codifica ha un insieme di cifre (**dizionario**)  $A$ 
  - In base 16, il dizionario è  $A_{16} = \{0, \dots, 9, A, \dots, F\}$
  - In base 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$
  - In base 8, il dizionario è  $A_8 = \{0, \dots, 7\}$
  - In base 2, il dizionario è  $A_2 = \{0, 1\}$
- Un numero è una sequenza di cifre
$$a_n a_{n-1} \dots a_1 a_0 \text{ con } a_i \in A$$
  - 8522 è una sequenza di 4 cifre di  $A_{10}$ ,  $\{8, 5, 2\} \subset A_{10}$
  - 4F è una sequenza di 2 cifre di  $A_{16}$ ,  $\{4, F\} \subset A_{16}$ .



## Codifica dei Numeri in una Base Qualsiasi

- Manteniamo un **codifica posizionale**: ogni cifra assume un significato diverso in base alla sua posizione nel numero.
  - $a_n$  è la cifra **più significativa**
  - $a_0$  è la cifra **meno significativa**

*Es:* in **8522**, **8** è la cifra più significativa, **2** quella meno.  
8522 è diverso da 2852, 8252,... che pur contengono le stesse cifre



## Codifica dei Numeri: Notazione Posizionale

- Dato un numero  $N_{10}$ , in base 10 contenente  $m$  cifre scritto come  $a_{m-1}a_{m-2} \dots a_1a_0$  questo corrisponde a:

$$N_{10} = a_{m-1} \times 10^{m-1} + a_{m-2} \times 10^{m-2} + \dots + a_0 \times 10^0$$



## Codifica dei Numeri: Notazione Posizionale

- Dato un numero  $N_{10}$ , in base 10 contenente  $m$  cifre scritto come  $a_{m-1}a_{m-2} \dots a_1a_0$  questo corrisponde a:

$$N_{10} = a_{m-1} \times 10^{m-1} + a_{m-2} \times 10^{m-2} + \dots + a_0 \times 10^0$$

$$(a_{m-1}a_{m-2} \dots a_1a_0)_{10} = \sum_{i=0}^{m-1} a_i \times 10^i, \quad a_i \in A_{10}$$

$$Es: (8522)_{10} = 8 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 2 \times 10^0$$



## Codifica dei Numeri: Notazione Posizionale

- Dato un numero  $N_{10}$ , in base 10 contenente  $m$  cifre scritto come  $a_{m-1}a_{m-2} \dots a_1a_0$  questo corrisponde a:

$$N_{10} = a_{m-1} \times 10^{m-1} + a_{m-2} \times 10^{m-2} + \dots + a_0 \times 10^0$$
$$(a_{m-1}a_{m-2} \dots a_1a_0)_{10} = \sum_{i=0}^{m-1} a_i \times 10^i, \quad a_i \in A_{10}$$

$$Es: (8522)_{10} = 8 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 2 \times 10^0$$

- Con  $m$  cifre in  $A_{10}$  quanti numeri posso esprimere:  $10^m$
- Considerando gli interi positivi, posso scrivere tutti numeri tra  $[0, 10^m - 1]$ 
  - *Es:*  $m = 1$  copro  $[0, 10 - 1]$  (cioè  $[0, 9]$ )  
 $m = 3$  copro  $[0, 10^3 - 1]$  (cioè  $[0, 999]$ )



## Rappresentazioni Posizionali in Base $p$

- Consideriamo **rappresentazioni posizionali in base  $p$**  (con  $p > 0$ ) e chiamiamo  $A_p$  il dizionario di  $p$  cifre:
  - se  $p \leq 10$  prendiamo le cifre di  $A_{10}$ ,  $A_p = \{0, \dots, p - 1\}$





## Rappresentazioni Posizionali in Base $p$

- Consideriamo **rappresentazioni posizionali in base  $p$**  (con  $p > 0$ ) e chiamiamo  $A_p$  il dizionario di  $p$  cifre:
  - se  $p \leq 10$  prendiamo le cifre di  $A_{10}$ ,  $A_p = \{0, \dots, p - 1\}$
  - se  $p > 10$  aggiungiamo simboli  $A_p = \{0, \dots, 9, A, B, \dots\}$



## Rappresentazioni Posizionali in Base $p$

- Consideriamo **rappresentazioni posizionali in base  $p$**  (con  $p > 0$ ) e chiamiamo  $A_p$  il dizionario di  $p$  cifre:
  - se  $p \leq 10$  prendiamo le cifre di  $A_{10}$ ,  $A_p = \{0, \dots, p-1\}$
  - se  $p > 10$  aggiungiamo simboli  $A_p = \{0, \dots, 9, A, B, \dots\}$
- Un numero di  $m$  cifre in base  $p$ :

$$N_p = a_{m-1} \times p^{m-1} + a_{m-2} \times p^{m-2} + \dots + a_0 \times p^0$$
$$N_p = a_{m-1} a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times p^i, \quad a_i \in A_p$$



## Rappresentazioni Posizionali in Base $p$

- Consideriamo **rappresentazioni posizionali in base  $p$**  (con  $p > 0$ ) e chiamiamo  $A_p$  il dizionario di  $p$  cifre:
  - se  $p \leq 10$  prendiamo le cifre di  $A_{10}$ ,  $A_p = \{0, \dots, p - 1\}$
  - se  $p > 10$  aggiungiamo simboli  $A_p = \{0, \dots, 9, A, B, \dots\}$

- Un numero di  $m$  cifre in base  $p$ :

$$N_p = a_{m-1} \times p^{m-1} + a_{m-2} \times p^{m-2} + \dots + a_0 \times p^0$$
$$N_p = a_{m-1} a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times p^i, \quad a_i \in A_p$$

- Con  $m$  cifre in  $A_p$  quanti numeri posso esprimere:  $p^m$
- Considerando gli interi positivi, posso scrivere tutti numeri tra  $[0, p^m - 1]$



## Codifica dei numeri in base $p$ : Esempi

- *Es:*  $m = 1$  e  $p = 7$ , copro  $[0, 7 - 1]$  (cioè  $[0, 6]$ )  
 $m = 4$  e  $p = 7$ , copro  $[0, 7^4 - 1]$  (cioè  $[0, 2400]$ )  
 $m = 1$  e  $p = 13$ , copro  $[0, 13 - 1]$  (cioè  $[0, 12]$ )  
 $m = 4$  e  $p = 13$ , copro  $[0, 13^4 - 1]$  (cioè  $[0, 28560]$ )



## Codifica dei numeri in base $p$ : Esempi

- Es:  $m = 1$  e  $p = 7$ , copro  $[0, 7 - 1]$  (cioè  $[0, 6]$ )  
 $m = 4$  e  $p = 7$ , copro  $[0, 7^4 - 1]$  (cioè  $[0, 2400]$ )  
 $m = 1$  e  $p = 13$ , copro  $[0, 13 - 1]$  (cioè  $[0, 12]$ )  
 $m = 4$  e  $p = 13$ , copro  $[0, 13^4 - 1]$  (cioè  $[0, 28560]$ )

Al crescere di  $p$  cresce il «potere espressivo» del dizionario (con lo stesso numero di cifre posso scrivere molti più numeri).



## Codifica dei Numeri in Base 2

- I calcolatori sono in grado di operare con informazioni **binarie**. Quindi  $p = 2$  e  $A_2 = \{0, 1\}$

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

$$N_2 = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 2^i, \quad a_i \in \{0, 1\}$$



## Codifica dei Numeri in Base 2

- I calcolatori sono in grado di operare con informazioni **binarie**. Quindi  $p = 2$  e  $A_2 = \{0, 1\}$

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

$$N_2 = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 2^i, \quad a_i \in \{0, 1\}$$

- Un bit (*binary digit*) assume valore 0/1 corrispondente ad un determinato *stato fisico* (alta o bassa tensione nella cella di memoria)



## Codifica dei Numeri in Base 2

- I calcolatori sono in grado di operare con informazioni **binarie**. Quindi  $p = 2$  e  $A_2 = \{0, 1\}$

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

$$N_2 = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 2^i, \quad a_i \in \{0, 1\}$$

- Un bit (*binary digit*) assume valore 0/1 corrispondente ad un determinato *stato fisico* (alta o bassa tensione nella cella di memoria)
- Con  $m$  bit posso scrivere  $2^m$  numeri diversi, ad esempio tutti gli interi nell'intervallo  $[0, 2^m - 1]$





## Codifica dei Numeri in Base 2

- I calcolatori sono in grado di operare con informazioni **binarie**. Quindi  $p = 2$  e  $A_2 = \{0, 1\}$

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$
$$N_2 = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 2^i, \quad a_i \in \{0,1\}$$

- Un bit (*binary digit*) assume valore 0/1 corrispondente ad un determinato *stato fisico* (alta o bassa tensione nella cella di memoria)
- Con  $m$  bit posso scrivere  $2^m$  numeri diversi, ad esempio tutti gli interi nell'intervallo  $[0, 2^m - 1]$
- Il byte è una sequenza di 8 bit ed esprime  $2^8 = 256$  numeri diversi (ad esempio gli interi in  $[0, 255]$ )

00000000, 00000001, 00000010, ..., 11111111



## Conversione Binario ➡ Decimale

Utilizziamo la definizione di numero in notazione posizionale

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

*Es.*

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$$

$$(1100010)_2 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2 = (98)_{10}$$




## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:



## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:
  - $531 / 2 = 265 + 1$   Divisione intera tra il numero e 2



## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:

- $531 / 2 = 265 + 1$

- $265 / 2 = 132 + 1$

Divisione intera tra il numero e 2

Il risultato della divisione precedente viene successivamente diviso



## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:

- $531 / 2 = 265 + 1$

- $265 / 2 = 132 + 1$

- $132 / 2 = 66 + 0$

Divisione intera tra il numero e 2

Il risultato della divisione precedente viene successivamente diviso



## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:

- $531 / 2 = 265 + 1$

Divisione intera tra il numero e 2

- $265 / 2 = 132 + 1$

Il risultato della divisione precedente viene successivamente diviso

- $132 / 2 = 66 + 0$

- $66 / 2 = 33 + 0$

- $33 / 2 = 16 + 1$

- $16 / 2 = 8 + 0$

- $8 / 2 = 4 + 0$

- $4 / 2 = 2 + 0$

- $2 / 2 = 1 + 0$

- $1 / 2 = 0 + 1$

Si continua fino a quando il risultato della divisione non diventa 0 (e considero comunque il resto!)



## Conversione Decimale ➡ Binario

- Metodo delle divisioni successive:
- Per convertire 531 opero come segue:

•	531	/	2	=	265	+	1
•	265	/	2	=	132	+	1
•	132	/	2	=	66	+	0
•	66	/	2	=	33	+	0
•	33	/	2	=	16	+	1
•	16	/	2	=	8	+	0
•	8	/	2	=	4	+	0
•	4	/	2	=	2	+	0
•	2	/	2	=	1	+	0
•	1	/	2	=	0	+	1

➡ Cifra meno significativa

I resti della divisione  
intera, letti dall'ultimo  
al primo, identificano  
la codifica binaria

$$(531)_{10} = (1000010011)_2$$

➡ Cifra più significativa





## Conversione Decimale ➡ Binario

- L'algoritmo delle divisioni successive vale rispetto a qualunque base



## Conversioni ottale/esadecimale ➡ decimale

- È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 16^i, \quad a_i \in A_{16}$$

- Es:  $(31)_8 =$

- $(A170)_{16} =$   
 $=$

- $(623)_8 =$   
 $=$

- $(623)_{16} =$   
 $=$



## Conversioni ottale/esadecimale ➡ decimale

- È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 16^i, \quad a_i \in A_{16}$$

- Es:  $(31)_8 = 3 * 8 + 1 = (25)_{10}$

- $(A170)_{16} =$   
 $=$

- $(623)_8 =$   
 $=$

- $(623)_{16} =$   
 $=$



## Conversioni ottale/esadecimale ➡ decimale

- È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 16^i, \quad a_i \in A_{16}$$

- Es:  $(31)_8 = 3 * 8 + 1 = (25)_{10}$
- $(A170)_{16} = A * 16^3 + 1 * 16^2 + 7 * 16$   
 $= 10 * 4096 + 1 * 256 + 7 * 16 = (41328)_{10}$
- $(623)_8 =$   
 $=$
- $(623)_{16} =$   
 $=$



## Conversioni ottale/esadecimale ➡ decimale

- È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 16^i, \quad a_i \in A_{16}$$

- Es:  $(31)_8 = 3 * 8 + 1 = (25)_{10}$
- $(A170)_{16} = A * 16^3 + 1 * 16^2 + 7 * 16$   
 $= 10 * 4096 + 1 * 256 + 7 * 16 = (41328)_{10}$
- $(623)_8 = 6 * 8^2 + 2 * 8 + 3 * 8^0$   
 $= 6 * 64 + 16 + 3 = (403)_{10}$
- $(623)_{16} =$   
 $=$



## Conversioni ottale/esadecimale ➡ decimale

- È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1a_0 = \sum_{i=0}^{m-1} a_i \times 16^i, \quad a_i \in A_{16}$$

- Es:  $(31)_8 = 3 * 8 + 1 = (25)_{10}$
- $(A170)_{16} = A * 16^3 + 1 * 16^2 + 7 * 16$   
 $= 10 * 4096 + 1 * 256 + 7 * 16 = (41328)_{10}$
- $(623)_8 = 6 * 8^2 + 2 * 8 + 3 * 8^0$   
 $= 6 * 64 + 16 + 3 = (403)_{10}$
- $(623)_{16} = 6 * 16^2 + 2 * 16 + 3 * 16^0$   
 $= 6 * 256 + 32 + 3 = (1571)_{10}$



## Conversioni decimale ➡ ottale/esadecimale

- È possibile utilizzare l'algoritmo delle divisioni successive



## Conversioni decimale ➡ ottale/esadecimale

- È possibile utilizzare l'algoritmo delle divisioni successive
- È tuttavia più comodo fare delle conversioni passando dalla rappresentazione binaria e





## Conversioni decimale ➡ ottale/esadecimale

- È possibile utilizzare l'algoritmo delle divisioni successive
- È tuttavia più comodo fare delle conversioni passando dalla rappresentazione binaria e
  - Esprimere ogni sequenza di 3 numeri binari in base 8
  - $(1231)_{10} = (10011001111)_2 = (\underbrace{010}_{2} \underbrace{011}_{3} \underbrace{001}_{1} \underbrace{111}_{7})_2$
  - $(1231)_{10} = (2317)_8$   
 $(2 \quad 3 \quad 1 \quad 7)_8$



## Conversioni decimale ➡ ottale/esadecimale

- È possibile utilizzare l'algoritmo delle divisioni successive
- È tuttavia più comodo fare delle conversioni passando dalla rappresentazione binaria e
  - Esprimere ogni sequenza di 3 numeri binari in base 8
  - $(1231)_{10} = (10011001111)_2 = (\underbrace{010}_{2} \underbrace{011}_{3} \underbrace{001}_{1} \underbrace{111}_{7})_8$
  - $(1231)_{10} = (2317)_8$
  - Esprimere ogni sequenza di 4 numeri binari in base 16
  - $(1231)_{10} = (10011001111)_2 = (\underbrace{0100}_{4} \underbrace{1100}_{C} \underbrace{1111}_{F})_{16}$
  - $(1231)_{10} = (4CF)_{16}$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline \phantom{0101}0 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 10 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 110 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$





## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline 1 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} \textcolor{red}{1} \\ 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline 01 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} \textcolor{red}{11} \\ 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline 001 \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} \textcolor{red}{111} \\ 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline 1001 \quad \textcolor{red}{(9)_{10}} \end{array}$$



## Somma tra Numeri Binari

- Si eseguono «in colonna» e si opera cifra per cifra
- Si considera il riporto come per i decimali
  - $0 + 0 = 0$  riporto 0
  - $1 + 0 = 1$  riporto 0
  - $0 + 1 = 1$  riporto 0
  - $1 + 1 = 0$  riporto 1
- Occorre sommare il riporto della cifra precedente

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} \textcolor{red}{111} \\ 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline \textcolor{red}{(1)}1001 \quad (25)_{10} \end{array}$$

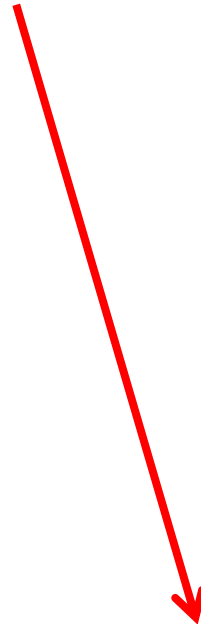


## Somma tra Numeri Binari

- A volte i bit utilizzati per codificare gli addendi non bastano a contenere il risultato
  - In questi casi occorrono più bit per codificare il risultato
  - Si ha quindi un bit di **carry in eccesso** e di conseguenza **overflow**

$$\begin{array}{r} \textcolor{red}{1} \\ 0101 + (5)_{10} \\ 1001 = (9)_{10} \\ \hline 1110 \quad (14)_{10} \end{array}$$

$$\begin{array}{r} \textcolor{red}{111} \\ 1111 + (15)_{10} \\ 1010 = (10)_{10} \\ \hline \textcolor{red}{(1)}1001 \quad (25)_{10} \end{array}$$





# Rappresentazione dei numeri interi

- Positivi e Negativi





## Rappresentazione Modulo e Segno

- È possibile dedicare il **primo bit** alla codifica del **segno**
  - "1" il numero che segue è negativo
  - "0" il numero che segue è positivo



## Rappresentazione Modulo e Segno

- È possibile dedicare il **primo bit** alla codifica del **segno**
  - "1" il numero che segue è negativo
  - "0" il numero che segue è positivo
- Con  $m$  cifre in binario e codifica modulo e segno dedico  $2^{m-1}$  rappresentazioni per i positivi e  $2^{m-1}$  rappresentazioni per gli stessi numeri cambiati di segno
  - posso rappresentare tutti i numeri nell'intervallo

$$X \in [-(2^{m-1} - 1), 2^{m-1} - 1]$$



## Rappresentazione Modulo e Segno

- È possibile dedicare il **primo bit** alla codifica del **segno**
  - "1" il numero che segue è negativo
  - "0" il numero che segue è positivo
- *Es su 5 bit rappresento*
  - $11111 = -15$
  - $10000 = -0$
  - $00000 = +0$
  - $01111 = +15$





**Ho due codifiche differenti lo zero**

- C'è uno «spreco» nella codifica



## Ho due codifiche differenti lo zero

- C'è uno «spreco» nella codifica
- Ostacola realizzazione circuitale delle operazioni algebriche
  - $A > 0 \ \&\& \ B > 0 \Rightarrow A + B == A + B$
  - $A > 0 \ \&\& \ B < 0 \Rightarrow A + B == A - |B|$
  - $A < 0 \ \&\& \ B > 0 \Rightarrow A + B == B - |A|$
  - $A < 0 \ \&\& \ B < 0 \Rightarrow A + B == -(|A| + |B|)$



## Ho due codifiche differenti lo zero

- C'è uno «spreco» nella codifica
- Ostacola realizzazione circuitale delle operazioni algebriche
  - $A > 0 \ \&\& \ B > 0 \Rightarrow A + B == A + B$
  - $A > 0 \ \&\& \ B < 0 \Rightarrow A + B == A - |B|$
  - $A < 0 \ \&\& \ B > 0 \Rightarrow A + B == B - |A|$
  - $A < 0 \ \&\& \ B < 0 \Rightarrow A + B == -(|A| + |B|)$
- Occorre trovare una rappresentazione migliore!



# Rappresentazione in complemento a 2 (CP2)



## Rappresentazione in Complemento a 2 (CP2)

- Date  $m$  cifre binarie, disponibili  $2^m$  configurazioni distinte
- In CP2 se ne usano:
  - $2^{m-1} - 1$  per valori positivi
  - 1 per lo zero
  - $2^{m-1}$  per i valori negativi
- Con  $m$  bit rappresento l'intervallo  $[-2^{m-1}, 2^{m-1} - 1]$





## Rappresentazione in Complemento a 2 (CP2)

- Sia  $X \in [-2^{m-1}, 2^{m-1} - 1]$  il numero da rappresentare in CP2, con  $m$  bit.
  - se  $X$  è **positivo** o nullo **scrivo**  $X$  in binario con  **$m$  bit**
  - se  $X$  è **negativo** **scrivo**  $2^m - |X|$  in binario con  **$m$  bit**

## DEFINIZIONE DI CP2



## Rappresentazione in Complemento a 2 (CP2)

- Esempio  $m = 3 \Rightarrow 2^3 = 8$ 
  - $-4 = 2^3 - 4 = 4 = 100$
  - $-3 =$
  - $-2 =$
  - $-1 =$
  - $0 =$
  - $1 =$
  - $2 =$
  - $3 =$



## Rappresentazione in Complemento a 2 (CP2)

- Esempio  $m = 3 \Rightarrow 2^3 = 8$ 
  - $-4 = 2^3 - 4 = 4 = 100$
  - $-3 = 2^3 - 3 = 5 = 101$
  - $-2 =$
  - $-1 =$
  - $0 =$
  - $1 =$
  - $2 =$
  - $3 =$



## Rappresentazione in Complemento a 2 (CP2)

- Esempio  $m = 3 \Rightarrow 2^3 = 8$ 
  - $-4 = 2^3 - 4 = 4 = 100$
  - $-3 = 2^3 - 3 = 5 = 101$
  - $-2 = 2^3 - 2 = 6 = 110$
  - $-1 =$
  - $0 =$
  - $1 =$
  - $2 =$
  - $3 =$



## Rappresentazione in Complemento a 2 (CP2)

- Esempio  $m = 3 \Rightarrow 2^3 = 8$ 
  - $-4 = 2^3 - 4 = 4 = 100$
  - $-3 = 2^3 - 3 = 5 = 101$
  - $-2 = 2^3 - 2 = 6 = 110$
  - $-1 = 2^3 - 1 = 7 = 111$
  - $0 =$
  - $1 =$
  - $2 =$
  - $3 =$



## Rappresentazione in Complemento a 2 (CP2)

- Esempio  $m = 3 \Rightarrow 2^3 = 8$ 
  - $-4 = 2^3 - 4 = 4 = 100$
  - $-3 = 2^3 - 3 = 5 = 101$
  - $-2 = 2^3 - 2 = 6 = 110$
  - $-1 = 2^3 - 1 = 7 = 111$
  - $0 = 000$
  - $1 = 001$
  - $2 = 010$
  - $3 = 011$



## Rappresentazione in CP2

- Con i positivi copro solo il range  $[0, 2^{m-1}-1]$ , quindi la prima cifra è 0



## Rappresentazione in CP2

- Con i positivi copro solo il range  $[0, 2^{m-1}-1]$ , quindi la prima cifra è 0
- Con i negativi copro il range  $[-2^{m-1}, -1]$  e scrivo  $2^m - |X|$ , e quindi la prima cifra è 1





## Rappresentazione in CP2

- Con i positivi copro solo il range  $[0, 2^{m-1}-1]$ , quindi la prima cifra è 0
- Con i negativi copro il range  $[-2^{m-1}, -1]$  e scrivo  $2^m - |X|$ , e quindi la prima cifra è 1
- Quindi, il primo bit **indica il segno** del numero
  - Attenzione: indica il segno ma non è il segno; **cambiandolo non** si ottiene il **numero opposto**
  - $45 = (0101101)_{CP2}$  se cambio di segno alla prima cifra
  - $(1101101)_{CP2} \rightarrow -2^6 + 2^5 + 2^3 + 2^2 + 1 = -64 + 45 = -19$



## Rappresentazione in CP2

- Con i positivi copro solo il range  $[0, 2^{m-1}-1]$ , quindi la prima cifra è 0
- Con i negativi copro il range  $[-2^{m-1}, -1]$  e scrivo  $2^m - |X|$ , e quindi la prima cifra è 1
- Quindi, il primo bit **indica il segno** del numero
  - Attenzione: indica il segno ma non è il segno; **cambiandolo non si ottiene il numero opposto**
  - $45 = (0101101)_{CP2}$  se cambio di segno alla prima cifra
  - $(1101101)_{CP2} \rightarrow -2^6 + 2^5 + 2^3 + 2^2 + 1 = -64 + 45 = -19$
- Inoltre, un solo valore per lo 0 nessuna configurazione “sprecata” dalla codifica



## CP2: rappresentazione formale

Dato un numero in CP2, il suo valore in base 10 è:

$$\begin{aligned} N_{CP2} &= a_{m-1}a_{m-2} \dots a_1a_0 \\ &= -a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0 \\ &= -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i, \quad a_i \in \{0,1\} \end{aligned}$$



## CP2: rappresentazione formale

Dato un numero in CP2, il suo valore in base 10 è:

$$\begin{aligned} N_{CP2} &= a_{m-1}a_{m-2} \dots a_1a_0 \\ &= \boxed{-a_{m-1} \times 2^{m-1}} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0 \\ &= \boxed{-a_{m-1} \times 2^{m-1}} + \sum_{i=0}^{m-2} a_i \times 2^i, \quad a_i \in \{0,1\} \end{aligned}$$

viene cambiato il segno dell'addendo relativo alla cifra più significativa



## CP2: rappresentazione formale

Esempi:

$$100 = -1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = -4$$



## CP2: rappresentazione formale

Esempi:

$$100 = -1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = -4$$

$$110 = -1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = -2$$



## CP2: rappresentazione formale

Esempi:

$$100 = -1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = -4$$

$$110 = -1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = -2$$

$$010 = -0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = +2$$



## Rappresentazione in CP2

- *Es, definire un intervallo che contenga -23 e 45*





## Rappresentazione in CP2

- *Es, definire un intervallo che contenga -23 e 45*
  - $m = 7$ , copro  $[-2^6, 2^6 - 1] = [-64, 63]$



## Rappresentazione in CP2

- *Es, definire un intervallo che contenga -23 e 45*
  - $m = 7$ , copro  $[-2^6, 2^6 - 1] = [-64, 63]$
  - ~~$m = 6$ , copro  $[-2^5, 2^5 - 1] = [-32, 31]$  (non cont. 45)~~



## Rappresentazione in CP2

- *Es, definire un intervallo che contenga -23 e 45*
  - $m = 7$ , copro  $[-2^6, 2^6 - 1] = [-64, 63]$
  - ~~$m = 6$ , copro  $[-2^5, 2^5 - 1] = [-32, 31]$  (non cont. 45)~~
- $-23 \rightarrow 2^7 - 23 = 128 - 23 = 105 = (1101001)_{CP2}$



## Rappresentazione in CP2

- *Es, definire un intervallo che contenga -23 e 45*
  - $m = 7$ , copro  $[-2^6, 2^6 - 1] = [-64, 63]$
  - ~~$m = 6$ , copro  $[-2^5, 2^5 - 1] = [-32, 31]$  (non cont. 45)~~
- $-23 \rightarrow 2^7 - 23 = 128 - 23 = 105 = (1101001)_{CP2}$
- $45 = (0101101)_{CP2}$



## Conversione Decimale ➡ CP2

Metodo "operativo" per rappresentare  $X$  su  $m$  bit in CP2



## Conversione Decimale ➔ CP2

Metodo "operativo" per rappresentare  $X$  su  $m$  bit in CP2

1. Controllo che  $X \in [-2^{m-1}, 2^{m-1} - 1]$ , altrimenti  $m$  bit non bastano



## Conversione Decimale ➡ CP2

Metodo "operativo" per rappresentare  $X$  su  $m$  bit in CP2

1. Controllo che  $X \in [-2^{m-1}, 2^{m-1} - 1]$ , altrimenti  $m$  bit non bastano
2. Se  $X$  è positivo, converto (col metodo delle divisioni successive)  $X$  utilizzando  $m$  bit  
**NB:** ricordandosi di aggiungerei zeri se necessario all'inizio del numero!



## Conversione Decimale ➔ CP2

Metodo "operativo" per rappresentare  $X$  su  $m$  bit in CP2

1. Controllo che  $X \in [-2^{m-1}, 2^{m-1} - 1]$ , altrimenti  $m$  bit non bastano

2. Se  $X$  è positivo, converto (col metodo delle divisioni successive)  $X$  utilizzando  $m$  bit

**NB:** ricordandosi di aggiungerei zeri se necessario all'inizio del numero!

3. Se  $X$  è negativo:

a) Converto (col metodo delle divisioni successive)  $|X|$  utilizzando  $m$  bit

b) **Complemento** tutti i bit di  $X$  ( $1 \rightarrow 0, 0 \rightarrow 1$ )

c) **Sommo 1** al numero ottenuto





## Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari



## Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari

i.  $m = 7$  copre  $[-2^6, 2^6 - 1] = [-64, 63]$



## Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari

i.  $m = 7$  copre  $[-2^6, 2^6 - 1] = [-64, 63]$

ii. Scrivo  $(56)_{10} \rightarrow 0111000$

56	0
28	0
14	0
7	1
3	1
1	1
0	



## Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari

- i.  $m = 7$  copre  $[-2^6, 2^6 - 1] = [-64, 63]$
- ii. Scrivo  $(56)_{10} \rightarrow 0111000$
- iii. Complemento  $\rightarrow 1000111$

56	0
28	0
14	0
7	1
3	1
1	1
0	

## Esempi Conversione Decimale ➡ CP2

## Esempio: scrivere -56 in CP2 con il numero di bit necessari

- i.  $m = 7$  copre  $[-2^6, 2^6 - 1] = [-64, 63]$
- ii. Scrivo  $(56)_{10} \rightarrow 0111000$
- iii. Complemento  $\rightarrow 1000111$
- iv. Sommo 1                                  1
- v.  $(1001000)_{CP2} = (-56)_{10}$

56	0
28	0
14	0
7	1
3	1
1	1
0	

## Esempi Conversione Decimale ➡ CP2

## Esempio: scrivere -56 in CP2 con il numero di bit necessari

- i.  $m = 7$  copre  $[-2^6, 2^6 - 1] = [-64, 63]$
- ii. Scrivo  $(56)_{10} \rightarrow 0111000$
- iii. Complemento  $\rightarrow 1000111$
- iv. Sommo 1                      1
- v.  $(1001000)_{CP2} = (-56)_{10}$

56	0
28	0
14	0
7	1
3	1
1	1
0	

Oppure applico la definizione:

$$(-56)_{10} = 2^7 - 56 = 72 = (1001000)_{CP2}$$



## Conversione CP2 ➡ Decimale

Possiamo utilizzare la definizione

$$\begin{aligned} N_{CP2} &= a_{m-1}a_{m-2} \dots a_1a_0 \\ &= -a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0 \\ &= -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i, \quad a_i \in \{0,1\} \end{aligned}$$



## Conversione CP2 ➡ Decimale

Possiamo utilizzare la definizione

$$\begin{aligned} N_{CP2} &= a_{m-1}a_{m-2} \dots a_1a_0 \\ &= -a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0 \\ &= -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i, \quad a_i \in \{0,1\} \end{aligned}$$

$$Es (1001000)_{CP2} = -2^6 + 2^3 = -64 + 8 = (-56)_{10}$$





## Somma tra Numeri in CP2

- In CP2 l'operazione di somma si realizza **come nella rappresentazione binaria posizionale**



## Somma tra Numeri in CP2

- In CP2 l'operazione di somma si realizza **come nella rappresentazione binaria posizionale**
- Grazie alla rappresentazione in CP2 è **possibile eseguire** anche **sottrazioni** tra numeri binari con lo stesso meccanismo (i.e., somme tra interi di segno opposto)



## Carry e Overflow in CP2

- In CP2 occorre individuare l'**overflow**, i.e., casi in cui il risultato è fuori dall'intervallo rappresentabile con i bit utilizzati.



## Carry e Overflow in CP2

- In CP2 occorre individuare l'**overflow**, i.e., casi in cui il risultato è fuori dall'intervallo rappresentabile con i bit utilizzati.
  - Quando c'è **overflow il risultato è inconsistente** con gli addendi:
    - Somma di due addendi positivi da un numero negativo
    - Somma di due addendi negativi da un numero positivo



## Carry e Overflow in CP2

- In CP2 occorre individuare l'**overflow**, i.e., casi in cui il risultato è fuori dall'intervallo rappresentabile con i bit utilizzati.
  - Quando c'è **overflow il risultato è inconsistente** con gli addendi:
    - Somma di due addendi positivi da un numero negativo
    - Somma di due addendi negativi da un numero positivo
- **NB non può esserci overflow quando sommo due numeri di segno opposto**



## Carry e Overflow in CP2

- In CP2 occorre individuare l'**overflow**, i.e., casi in cui il risultato è fuori dall'intervallo rappresentabile con i bit utilizzati.
  - Quando c'è **overflow il risultato è inconsistente** con gli addendi:
    - Somma di due addendi positivi da un numero negativo
    - Somma di due addendi negativi da un numero positivo
- **NB** non può esserci overflow quando sommo due numeri di segno opposto
- **NB** può esserci un ultimo riporto senza che ci sia overflow e viceversa



## Esempio senza overflow

*Esempio:* 60 – 54



## Esempio senza overflow

*Esempio:*  $60 - 54$

diventa  $60 + (-54)$





## Esempio senza overflow

*Esempio:*  $60 - 54$

diventa  $60 + (-54)$

$$\begin{array}{rcl} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} \\ (60)_{10} & = & (0 & 1 & 1 & 1 & 1 & 0 & 0)_{CP2} \\ (-54)_{10} & = & (1 & 0 & 0 & 1 & 0 & 1 & 0)_{CP2} \\ & & \hline & (1) & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$



## Esempio senza overflow

*Esempio:*  $60 - 54$

diventa  $60 + (-54)$

$$\begin{array}{r} \phantom{(60)_{10} = } \phantom{= } \phantom{(0\ 1\ 1\ 1\ 1\ 0\ 0)_{CP2}} \phantom{(0\ 1\ 1\ 1\ 1\ 0\ 0)_{CP2}} \\ \phantom{(60)_{10} = } \phantom{= } \phantom{(0\ 1\ 1\ 1\ 1\ 0\ 0)_{CP2}} \phantom{(0\ 1\ 1\ 1\ 1\ 0\ 0)_{CP2}} \\ (60)_{10} = (0\ 1\ 1\ 1\ 1\ 0\ 0)_{CP2} \\ (-54)_{10} = (1\ 0\ 0\ 1\ 0\ 1\ 0)_{CP2} \\ \hline (1)\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \end{array}$$

Il riporto (carry) viene ignorato

Quando sommo numeri di segno opposto non può esserci overflow

Il risultato è positivo  $(0000110)_{CP2} = (6)_{10}$



## Esempio

$$\text{Esempio: } -4 - 3 = -4 + (-3) = (100)_{CP2} + (101)_{CP2}$$



## Esempio

Esempio:  $-4 - 3 = -4 + (-3) = (100)_{CP2} + (101)_{CP2}$

1

1 0 0 +

1 0 1 =

---

(1) 0 0 1



## Esempio

Esempio:  $-4 - 3 = -4 + (-3) = (100)_{CP2} + (101)_{CP2}$

1

1 0 0 +

1 0 1 =

---

(1) 0 0 1



- Ignoro il bit di carry



## Esempio

Esempio:  $-4 - 3 = -4 + (-3) = (100)_{CP2} + (101)_{CP2}$

1

1 0 0 +

1 0 1 =

---

(1) 0 0 1



- Ignoro il bit di carry
- **Overflow:** la somma di due numeri negativi mi ha dato un numero positivo.
- Il risultato non ha senso, occorre scrivere gli addendi con un bit in più per rappresentare il risultato dell'operazione



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{r} -3 \Rightarrow \\ -4 \Rightarrow \\ \hline -7 \Rightarrow \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \\ +6 \Rightarrow \\ \hline +3 \Rightarrow \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \\ -7 \Rightarrow \\ \hline -10 \Rightarrow \end{array}$$

$$\begin{array}{r} +2 \Rightarrow \\ +5 \Rightarrow \\ \hline +7 \Rightarrow \end{array}$$

$$\begin{array}{r} +3 \Rightarrow \\ +6 \Rightarrow \\ \hline +9 \Rightarrow \end{array}$$



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -4 & \Rightarrow & 1100 \\ \hline -7 & \Rightarrow & (1)1001 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & \\ +6 & \Rightarrow & \\ \hline +3 & \Rightarrow & \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & \\ -7 & \Rightarrow & \\ \hline -10 & \Rightarrow & \end{array}$$

$$\begin{array}{rcl} +2 & \Rightarrow & \\ +5 & \Rightarrow & \\ \hline +7 & \Rightarrow & \end{array}$$

$$\begin{array}{rcl} +3 & \Rightarrow & \\ +6 & \Rightarrow & \\ \hline +9 & \Rightarrow & \end{array}$$





## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$-3 \Rightarrow 1101$$

$$\begin{array}{r} -4 \Rightarrow 1100 \\ \hline \end{array}$$

$$-7 \Rightarrow (1)1001$$

$$-3 \Rightarrow 1101$$

$$\begin{array}{r} +6 \Rightarrow 0110 \\ \hline \end{array}$$

$$+3 \Rightarrow (1)0011$$

$$-3 \Rightarrow$$

$$\begin{array}{r} -7 \Rightarrow \\ \hline \end{array}$$

$$-10 \Rightarrow$$

$$+2 \Rightarrow$$

$$\begin{array}{r} +5 \Rightarrow \\ \hline \end{array}$$

$$+7 \Rightarrow$$

$$+3 \Rightarrow$$

$$\begin{array}{r} +6 \Rightarrow \\ \hline \end{array}$$

$$+9 \Rightarrow$$



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -4 & \Rightarrow & 1100 \\ \hline -7 & \Rightarrow & (1)1001 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ +6 & \Rightarrow & 0110 \\ \hline +3 & \Rightarrow & (1)0011 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -7 & \Rightarrow & 1001 \\ \hline -10 & \Rightarrow & (1)0110 \end{array}$$

$$\begin{array}{rcl} +2 & \Rightarrow & \\ +5 & \Rightarrow & \\ \hline +7 & \Rightarrow & \end{array}$$

$$\begin{array}{rcl} +3 & \Rightarrow & \\ +6 & \Rightarrow & \\ \hline +9 & \Rightarrow & \end{array}$$



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -4 & \Rightarrow & 1100 \\ \hline -7 & \Rightarrow & (1)1001 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ +6 & \Rightarrow & 0110 \\ \hline +3 & \Rightarrow & (1)0011 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -7 & \Rightarrow & 1001 \\ \hline -10 & \Rightarrow & (1)0110 \end{array}$$

$$\begin{array}{rcl} +2 & \Rightarrow & 0010 \\ +5 & \Rightarrow & 0101 \\ \hline +7 & \Rightarrow & (0)0111 \end{array}$$

$$\begin{array}{rcl} +3 & \Rightarrow & \\ +6 & \Rightarrow & \\ \hline +9 & \Rightarrow & \end{array}$$



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -4 & \Rightarrow & 1100 \\ \hline -7 & \Rightarrow & (1)1001 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ +6 & \Rightarrow & 0110 \\ \hline +3 & \Rightarrow & (1)0011 \end{array}$$

$$\begin{array}{rcl} -3 & \Rightarrow & 1101 \\ -7 & \Rightarrow & 1001 \\ \hline -10 & \Rightarrow & (1)0110 \end{array}$$

$$\begin{array}{rcl} +2 & \Rightarrow & 0010 \\ +5 & \Rightarrow & 0101 \\ \hline +7 & \Rightarrow & (0)0111 \end{array}$$

$$\begin{array}{rcl} +3 & \Rightarrow & 0011 \\ +6 & \Rightarrow & 0110 \\ \hline +9 & \Rightarrow & (0)1001 \end{array}$$



## Esempi

- Esempi: con  $m = 4$  bit  $\Rightarrow [-8, 7]$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad 1100 \\ \hline -7 \Rightarrow \quad (1)1001 \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ +6 \Rightarrow \quad 0110 \\ \hline +3 \Rightarrow \quad (1)0011 \end{array}$$

Non c'è alcuna relazione fra  
l'overflow e il carry!

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -7 \Rightarrow \quad 1001 \\ \hline -10 \Rightarrow \quad (1)0110 \end{array}$$

$$\begin{array}{r} +2 \Rightarrow \quad 0010 \\ +5 \Rightarrow \quad 0101 \\ \hline +7 \Rightarrow \quad (0)0111 \end{array}$$

$$\begin{array}{r} +3 \Rightarrow \quad 0011 \\ +6 \Rightarrow \quad 0110 \\ \hline +9 \Rightarrow \quad (0)1001 \end{array}$$



## Esempio

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.



## Esempio

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.
  - a)  $[-2^8, 2^8 - 1] = [-256, 255]$



## Esempio

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.
- a)  $[-2^8, 2^8 - 1] = [-256, 255]$
- b) Indicare, giustificando brevemente le risposte, quali delle seguenti operazioni possono essere effettuate :
- i.  $-254 - 255$
  - ii.  $+ 254 - 253$
  - iii.  $-18 + 236$
  - iv.  $+ 217 + 182$





## Esempio

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.
- a)  $[-2^8, 2^8 - 1] = [-256, 255]$
- b) Indicare, giustificando brevemente le risposte, quali delle seguenti operazioni possono essere effettuate :
- i.  $-254 - 255 = -509$  ... OUT OF RANGE
  - ii.  $+ 254 - 253 = 1$  ... IN RANGE
  - iii.  $-18 + 236 = 218$  ... IN RANGE
  - iv.  $+ 217 + 182 = 399$  ... OUT OF RANGE



## Esempio

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.
- a)  $[-2^8, 2^8 - 1] = [-256, 255]$
- b) Indicare, giustificando brevemente le risposte, quali delle seguenti operazioni possono essere effettuate:
- i.  $-254 - 255 = -509$  ... OUT OF RANGE
  - ii.  $+ 254 - 253 = 1$  ... IN RANGE
  - iii.  $-18 + 236 = 218$  ... IN RANGE
  - iv.  $+ 217 + 182 = 399$  ... OUT OF RANGE
- c) Mostrare in dettaglio come avviene il calcolo delle operazioni (i) e (ii), evidenziando il bit di riporto e il bit di overflow così ottenuti. (Il bit di overflow è pari ad 1 se si verifica overflow, 0 altrimenti.)



## Esempio

$$-254 = 512 - 254 = 258 = 100000010$$



## Esempio

$$-255 = ?$$

$$255 / 2 = 127 + 1$$

$$127 / 2 = 63 + 1$$

$$63 / 2 = 31 + 1$$

$$31 / 2 = 15 + 1$$

$$15 / 2 = 7 + 1$$

$$7 / 2 = 3 + 1$$

$$3 / 2 = 1 + 1$$

$$1 / 2 = 0 + 1$$

$$01111111 \Rightarrow 100000000 \Rightarrow 100000001$$



## Esempio

$$-254 = 512 - 254 = 258 = 100000010$$

$$-255 = 512 - 255 = 257 = 100000001$$

$$\begin{array}{r} 100000010 \quad (-254) \\ 100000001 \quad (-255) \\ \hline (1) 000000011 \quad (-509) \end{array}$$



## Esempio

011111110 (+254)

100000011 (-253)

---

[0] (1) 000000001 (+1)



# Rappresentazione dei numeri reali



## I numeri reali

- Un'approssimazione dei numeri reali in  $[0,1]$
- Si rappresentano anteponendo **0.** al numero

$$N_p = (0. a_{-1} a_{-2} \dots a_{-m})_p =$$

$$N_p = a_{-1} \times p^{-1} + a_{-2} \times p^{-2} \dots + a_{-m} \times p^{-m}$$

$$N_p = \sum_{i=1}^m a_i \times p^{-i}, \quad a_i \in A_p$$





## I numeri reali

- Un'approssimazione dei numeri reali in  $[0,1]$
- Si rappresentano anteponendo **0.** al numero

$$N_p = (0. a_{-1} a_{-2} \dots a_{-m})_p =$$

$$N_p = a_{-1} \times p^{-1} + a_{-2} \times p^{-2} \dots + a_{-m} \times p^{-m}$$

$$N_p = \sum_{i=1}^m a_i \times p^{-i}, \quad a_i \in A_p$$

- In base 10

$$0.586 = 5 \times 10^{-1} + 8 \times 10^{-2} + 6 \times 10^{-3}$$



## I numeri reali

- Un'approssimazione dei numeri reali in  $[0,1]$
- Si rappresentano anteponendo **0.** al numero

$$N_p = (0. a_{-1} a_{-2} \dots a_{-m})_p =$$

$$N_p = a_{-1} \times p^{-1} + a_{-2} \times p^{-2} \dots + a_{-m} \times p^{-m}$$

$$N_p = \sum_{i=1}^m a_i \times p^{-i}, \quad a_i \in A_p$$

- In base 10

$$0.586 = 5 \times 10^{-1} + 8 \times 10^{-2} + 6 \times 10^{-3}$$

- In base 2

$$(0.101)_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = \frac{1}{2} + \frac{1}{8} = (0.625)_{10}$$



## I numeri reali

- Date  $m$  cifre in base  $p = 2$ , posso rappresentare **un sottoinsieme** dell'intervallo continuo  $[0, 1 - 2^{-m}]$



## I numeri reali

- Date  $m$  cifre in base  $p = 2$ , posso rappresentare **un sottoinsieme** dell'intervallo continuo  $[0, 1 - 2^{-m}]$ 
  - Con una cifra binaria dopo la virgola rappresento l'intervallo  $[0, 0.5]$ 
    - $(x.0)_2 = (y.0)_{10}$
    - $(x.1)_2 = (y.5)_{10}$



## I numeri reali

- Date  $m$  cifre in base  $p = 2$ , posso rappresentare **un sottoinsieme** dell'intervallo continuo  $[0, 1 - 2^{-m}]$ 
  - Con una cifra binaria dopo la virgola rappresento l'intervallo  $[0, 0.5]$ 
    - $(x.0)_2 = (y.0)_{10}$
    - $(x.1)_2 = (y.5)_{10}$
  - Con due cifre binarie dopo la virgola rappresento l'intervallo  $[0, 0.75]$ 
    - $(x.00)_2 = (y.0)_{10}$
    - $(x.01)_2 = (y.25)_{10}$
    - $(x.10)_2 = (y.5)_{10}$
    - $(x.11)_2 = (y.75)_{10}$



## I numeri reali

- Date  $m$  cifre in base  $p = 2$ , posso rappresentare **un sottoinsieme** dell'intervallo continuo  $[0, 1 - 2^{-m}]$ 
  - Con una cifra binaria dopo la virgola rappresento l'intervallo  $[0, 0.5]$ 
    - $(x.0)_2 = (y.0)_{10}$
    - $(x.1)_2 = (y.5)_{10}$
  - Con due cifre binarie dopo la virgola rappresento l'intervallo  $[0, 0.75]$ 
    - $(x.00)_2 = (y.0)_{10}$
    - $(x.01)_2 = (y.25)_{10}$
    - $(x.10)_2 = (y.5)_{10}$
    - $(x.11)_2 = (y.75)_{10}$
- **L'errore di approssimazione** sarà minore di  $2^{-m}$



## Conversione Decimale ➡ Binario su Frazionari

L'algoritmo delle **moltiplicazioni successive** per convertire  $N_{10} \in [0,1]$

1. Moltiplico  $N_{10}$  per 2.
2. La parte intera del risultato definisce una cifra della rappresentazione binaria finale
3. la parte frazionaria risultante viene moltiplicata per 2
4. Si itera i passi 1- 3 fino a
  - Ottenere parte frazionaria nulla (rappresentazione esatta)  
**oppure**
  - Coprire tutte le cifre binarie a disposizione (rappresentazione approssimata)
5. **La cifra più significativa** (il coefficiente di  $2^{-1}$ ) è dato dalla **prima parte intera calcolata**, **quella meno significativa** (il coefficiente di  $2^{-m}$ ) è dato **dall'ultima parte intera calcolata**



## Esempio

- Convertire in binario 0.625 utilizzando  $m = 6$  bit
  - $0.625 \times 2 = 1 + 0.25$   $\longrightarrow$  Parte intera +  
parte frazionaria
  - $0.250 \times 2 = 0 + 0.5$
  - $0.500 \times 2 = 1 + 0$
  - 0  $\longrightarrow$  La parte intera definisce la  
rappresentazione binaria
- 
- Otteniamo  $(0.625)_{10} = 0.101$ , la rappresentazione è  
esatta





## Esempio

- Convertire in binario 0.625 utilizzando  $m = 6$  bit
  - $0.625 \times 2 = 1 + 0.25$
  - $0.250 \times 2 = 0 + 0.5$
  - $0.500 \times 2 = 1 + 0$
  - 0
  - Otteniamo  $(0.625)_{10} = 0.101$ , la rappresentazione è esatta
-



## Esempio

- Convertire in binario 0.625 utilizzando  $m = 6$  bit
- $0.625 \times 2 = 1 + 0.25$
- $0.250 \times 2 = 0 + 0.5$
- $0.500 \times 2 = 1 + 0$
- 0
  
- Otteniamo  $(0.625)_{10} = 0.101$ , la rappresentazione è esatta
- Devo usare 6 bit:  $(0.625)_{10} = 0.101000$



## Esempio

- Convertire in binario 0.587 utilizzando  $m = 6$  bit
- $0.587 \times 2 = 1 + 0.174$  → Parte intera + parte frazionaria
- $0.174 \times 2 = 0 + 0.348$
- $0.348 \times 2 = 0 + 0.696$
- $0.696 \times 2 = 1 + 0.392$
- $0.392 \times 2 = 0 + 0.784$
- $0.784 \times 2 = 1 + 0.560$  → La parte intera definisce la rappresentazione binaria
- Otteniamo  $(0.587)_{10} \approx 0.100101$
- Rappresentazione approssimata, la parte frazionaria finale non è 0. L'errore introdotto è minore di  $2^{-6}$ .



## Esempio

- Convertire in binario 0.9 utilizzando  $m = 16$  bit

$$\begin{array}{l} - 0.9 \times 2 = 1 + 0.8 \\ - 0.8 \times 2 = 1 + 0.6 \\ - 0.6 \times 2 = 1 + 0.2 \\ - 0.2 \times 2 = 0 + 0.4 \\ - 0.4 \times 2 = 0 + 0.8 \\ - 0.8 \times 2 = 1 + 0.6 \\ - 0.6 \times 2 = 1 + 0.2 \\ - 0.2 \times 2 = 0 + 0.4 \\ - 0.4 \times 2 = 0 + 0.8 \\ - 0.8 \times 2 = 1 + 0.6 \\ - \dots \end{array}$$

Rappresentazione  
**periodica!** Inutile  
procedere oltre

- Otteniamo  $(0.9)_{10} \approx 0.1110011001100110$  con accuratezza di almeno  $2^{-16}$ .



## Rappresentazione in Virgola Fissa

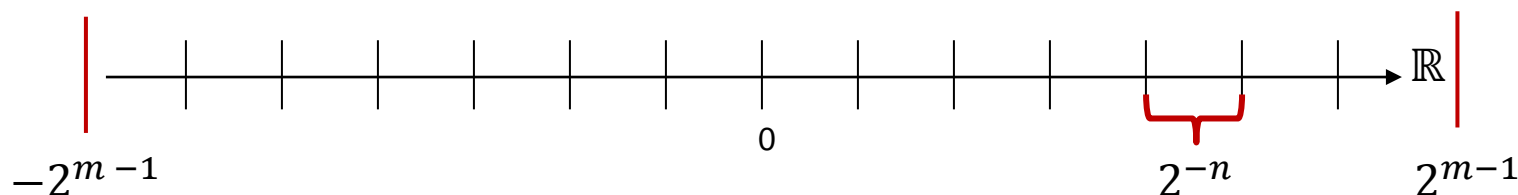
- Si definiscono  $m$  bit per la parte intera e  $n$  bit per la parte frazionaria e si scrivono le due parti indipendentemente
- Rappresentare  $(-123,21)_{10}$  utilizzando  $m = 8, n = 6$  e rappresentazione in  $CP_2$  per la parte intera

$$(-123)_{10} = (10000101)_{CP_2}$$

$$(0,21)_{10} \approx (001101)_2$$

$$-123,21_{10} \approx (10000101.001101)_2$$

- Questa rappresentazione mi da
  - Precisione costante lungo l'asse reale  $\mathbb{R}$ :
  - Estremi definiti solo da  $m$





## Virgola mobile (floating point)

- Il numero  $r$  in base  $p$  in virgola mobile è espresso come:

$$r = \pm M \cdot b^n$$

- $M$  mantissa (parte frazionaria, è un numero razionale)
- $b$ : base della notazione esponenziale (numero naturale)
- $n$ : esponente (numero intero)
- **$M$  e  $n$  sono in base  $p$**  (non necessariamente 10)



## Virgola mobile (floating point)

- Il numero  $r$  in base  $p$  in virgola mobile è espresso come:
$$r = \pm M \cdot b^n$$
  - $M$  mantissa (parte frazionaria, è un numero razionale)
  - $b$ : base della notazione esponenziale (numero naturale)
  - $n$ : esponente (numero intero)
  - **$M$  e  $n$  sono in base  $p$**  (non necessariamente 10)

*Esempio* ( $p = 10, b = 10$ ):

$$-331,6875 = -0,3316875 \cdot 10^3 \quad M = -0,3316875; \quad n = 3$$

- Il numero può essere codificato usando **un numero predefinito di bit** per  $M$  e per  $n$ 
  - $b$  e  $p$  non devono essere rappresentati, sono definiti dallo standard



## Virgola mobile (floating point)

- Il numero  $r$  in base  $p$  in virgola mobile è espresso come:
$$r = \pm M \cdot b^n$$
  - $M$  mantissa (parte frazionaria, è un numero razionale)
  - $b$ : base della notazione esponenziale (numero naturale)
  - $n$ : esponente (numero intero)
  - **$M$  e  $n$  sono in base  $p$**  (non necessariamente 10)
- **N.B** la base  $p$  della codifica può essere diversa dalla base della rappresentazione in floating point  $b$

*Esempio* Se ho ( $p = 2$   $b = 10$ ) e  $M = 1011$ ,  $n = 11 = 0.6875 \times 10^3$





## Normalizzazione

- A parità di precisione, il numero di cifre dopo la virgola può cambiare giocando sull'esponente  
e.g.:  $0,06789013245 \times 10^{15} = 6,789013245 \times 10^{13}$



## Normalizzazione

- A parità di precisione, il numero di cifre dopo la virgola può cambiare giocando sull'esponente  
e.g.:  $0,06789013245 \times 10^{15} = 6,789013245 \times 10^{13}$
- Numero in virgola mobile detto **normalizzato** se contiene una sola cifra nella parte intera  
**e.g.:**  $0,06789013245 \times 10^{15}$       NON normalizzato  
          $6,789013245 \times 10^{13}$         NORMALIZZATO



# Normalizzazione

- A parità di precisione, il numero di cifre dopo la virgola può cambiare giocando sull'esponente  
e.g.:  $0,06789013245 \times 10^{15} = 6,789013245 \times 10^{13}$
- Numero in virgola mobile detto **normalizzato** se contiene una sola cifra nella parte intera  
e.g.:  $0,06789013245 \times 10^{15}$       NON normalizzato  
          $6,789013245 \times 10^{13}$         NORMALIZZATO
- Forma normalizzata vantaggiosa se numero cifre (i.e., bit) disponibili per rappresentare  $r$  è limitato
  - permette di evitare zeri iniziali inutili

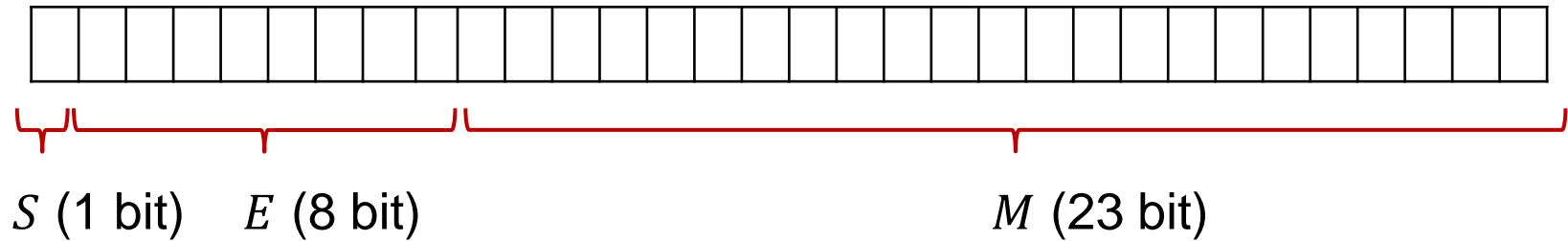


## Standard IEEE 754-1985

- Tre diversi formati, differiscono nel numero totale dei bit utilizzati. Quelli più diffusi:
  - precisione singola: 32 bit
  - precisione doppia: 64 bit
  - precisione estesa: 128 bit
- Si usa sempre base  $b = 2$  e  $p = 2$



## Standard IEEE 754-1985 a 32 bit

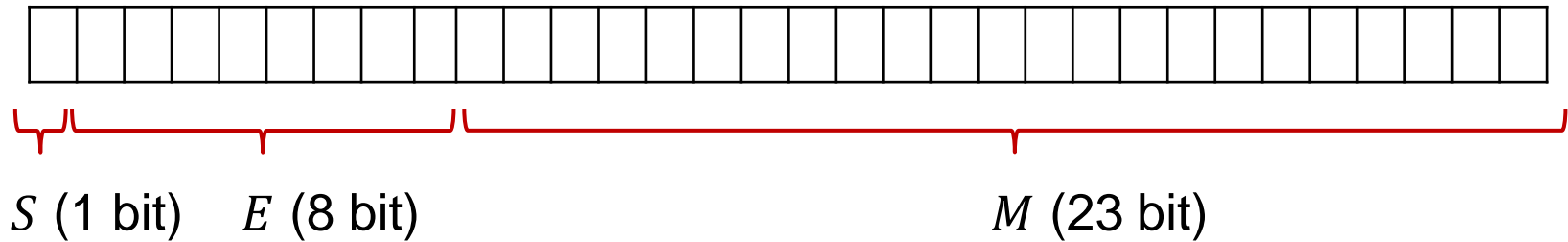


- Il numero  $X$  ha la seguente rappresentazione

$$X = (-1)^S \times 1.M \times 2^E$$



## Standard IEEE 754-1985 a 32 bit



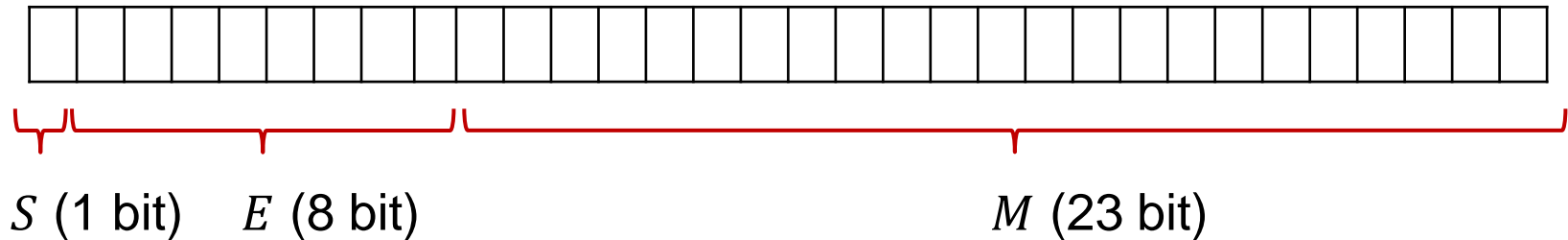
- Il numero  $X$  ha la seguente rappresentazione

$$X = (-1)^S \times 1.M \times 2^E$$

- Rappresentazione divisa in tre parti:
  - $S$ : il segno (1 bit)
  - $M$ : contiene le **cifre decimali** della *mantissa* in forma normalizzata (23 bit)
  - $E$ : l'esponente (8 bit)



## Standard IEEE 754-1985 a 32 bit



- Il numero  $X$  ha la seguente rappresentazione

$$X = (-1)^S \times 1.M \times 2^E$$

- Rappresentazione divisa in tre parti:
  - $S$ : il segno (1 bit)
  - $M$ : contiene le **cifre decimali** della *mantissa* in forma normalizzata (23 bit)
  - $E$ : l'esponente (8 bit)
- $S$ ,  $M$  ed  $E$  si rappresentano in base  $p = 2$ , la base della notazione esponenziale  $b = 2$ .



## Standard IEEE 754-1985, la mantissa ( $M$ )

- Rappresentata **in binario in forma normalizzata**,
- In base 2 la mantissa è un numero compreso tra  
 $1.00000 \dots 0$  e  $1.11111 \dots 1$
- Nella codifica IEEE 754-1985 in  $M$  **non viene mai salvata la prima cifra** che è sempre 1
$$X = (-1)^S \times 1.M \times 2^E$$
- **N.B.** la mantissa è sempre positiva, si usa il bit di segno  $S$  per rappresentare i numeri negativi.  $M$  **non** viene rappresentato in  $CP_2$  quindi.





## Standard IEEE 754-1985, l'esponente ( $E$ )

- Si usa una **notazione per eccesso**

$$E = n + 127.$$

Dove  $n$  è la caratteristica del numero, l'esponente di 2

- In questo modo **non occorre dedicare un bit al segno di  $E$** :
  - $n \in [-128, 127]$
  - voglio  $E \in [0, 255]$  ma con  $E = 0$  e  $E = 255$  casi speciali (vedremo dopo)
  - Di conseguenza per avere  $E \in [1, 254]$  ho bisogno che l'eccesso sia 127
  - E di conseguenza gli  $n$  ammissibili saranno  $\in [-126, 127]$



## Standard IEEE 754-1985, l'esponente ( $E$ )

- Si usa una **notazione per eccesso**

$$E = n + 127.$$

Dove  $n$  è la caratteristica del numero, l'esponente di 2

- Esempi:
  - esponente  $E = 254 \Rightarrow$  caratteristica  $n = +127$
  - esponente  $E = 250 \Rightarrow$  caratteristica  $n = +123$
  - esponente  $E = 132 \Rightarrow$  caratteristica  $n = +5$
  - esponente  $E = 127 \Rightarrow$  caratteristica  $n = 0$
  - esponente  $E = 80 \Rightarrow$  caratteristica  $n = -47$
  - esponente  $E = 1 \Rightarrow$  caratteristica  $n = -126$
- **N.B.**  $E > 127$  indica caratteristiche positive,  $E < 127$  indica caratteristiche negative



## Standard IEEE 754-1985, l'esponente ( $E$ )

- Si usa una **notazione per eccesso**

$$E = n + 127.$$

Dove  $n$  è la caratteristica del numero, l'esponente di 2

- Esempi:
  - esponente  $E = 254 \Rightarrow$  caratteristica  $n = +127$
  - esponente  $E = 250 \Rightarrow$  caratteristica  $n = +123$
  - esponente  $E = 132 \Rightarrow$  caratteristica  $n = +5$
  - esponente  $E = 127 \Rightarrow$  caratteristica  $n = 0$
  - esponente  $E = 80 \Rightarrow$  caratteristica  $n = -47$
  - esponente  $E = 1 \Rightarrow$  caratteristica  $n = -126$
- **N.B.**  $E$  va rappresentato **sempre con 8 bit**



## Come Procedere

Dato  $X = 42.6875$  numero reale:



## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)



## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)
2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$



## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)
2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$
3. Porto il numero in **forma normalizzata** in base 2
  - $X = 1.010101011 \times 2^5$



## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)
2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$
3. Porto il numero in **forma normalizzata** in base 2
  - $X = 1.010101011 \times 2^5$
4. Definisco  $M$  a 23 bit come la mantissa senza il primo bit (sempre 1)
  - $M = 01010101\ 10000000\ 0000000$  (23 bit)





## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)
2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$
3. Porto il numero in **forma normalizzata** in base 2
  - $X = 1.010101011 \times 2^5$
4. Definisco  $M$  a 23 bit come la mantissa senza il primo bit (sempre 1)
  - $M = 01010101\ 10000000\ 0000000$  (23 bit)
5. Calcolo  $E = n + 127 = 5 + 127 = 132 \rightarrow 10000100$  (8 bit)



## Come Procedere

Dato  $X = 42.6875$  numero reale:

1. Definisco il **bit di segno**  $S = 0$  se positivo,  $S = 1$  se negativo
  - $S = 0$  (1 bit)
2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$
3. Porto il numero in **forma normalizzata** in base 2
  - $X = 1.010101011 \times 2^5$
4. Definisco  $M$  a 23 bit come la mantissa senza il primo bit (sempre 1)
  - $M = 01010101\ 10000000\ 00000000$  (23 bit)
5. Calcolo  $E = n + 127 = 5 + 127 = 132 \rightarrow 10000100$  (8 bit)
6. Compongo il numero
  - $S = 0$
  - $E = 10000100$
  - $M = 01010101\ 10000000\ 00000000$



## Nota bene

2. **Codifico in virgola fissa in base 2**, parte frazionaria e parte intera
  - $X = 42.6875 \rightarrow 101010.1011 \times 2^0$
3. Porto il numero in **forma normalizzata** in base 2
  - $X = 1.010101011 \times 2^5$

**Non ha senso fare il viceversa:** normalizzare in base 10 e poi passare in base 2.

1. Il numero normalizzato in base 10 potrebbe non esserlo in base 2 (es. 9.5 -> 1001.01)



## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011 \ 0000000 \ 0000000$
- $E = 10000100$



## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011 \ 0000000 \ 0000000$
- $E = 10000100$
- $S = 0$  ...numero positivo



## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011 \ 0000000 \ 0000000$
- $E = 10000100$
  
- $S = 0$  ...numero positivo
- $E=10000100 \rightarrow E = 132 ; n = 132 - 127 = 5$



## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011\ 0000000\ 0000000$
- $E = 10000100$
- $S = 0$  ...numero positivo
- $E=10000100 \rightarrow E = 132 ; n = 132 - 127 = 5$
- $1.M\ (1.10010011\ 0000000\ 0000000) \rightarrow 1.10010011$



## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011\ 0000000\ 0000000$
- $E = 10000100$
- $S = 0$  ...numero positivo
- $E=10000100 \rightarrow E = 132 ; n = 132 - 127 = 5$
- $1.M$  ( $1.10010011\ 0000000\ 0000000$ )  $\rightarrow 1.10010011$
- Mantissa denormalizzata  $\rightarrow 110010.011$





## Esempio di Decodifica

Convertire in base dieci il seguente numero espresso nella codifica floating point:

- $S = 0$
- $M = 10010011 \ 0000000 \ 0000000$
- $E = 10000100$
- $S = 0$  ...numero positivo
- $E=10000100 \rightarrow E = 132 ; n = 132 - 127 = 5$
- $1.M$  ( $1.10010011 \ 0000000 \ 0000000$ )  $\rightarrow 1.10010011$
- Mantissa denormalizzata  $\rightarrow 110010.011$
- Parte intera:  $110010 = 32+16+2 = 50$
- Parte razionale:  $.011 = 0.25+0.125 = 0.375$
- $\rightarrow X = 50.375$



## Casi Particolari

Con la codifica IEEE 754-1985 è anche possibile scrivere:

- *NaN*: (Not a Number, valori non definiti):  $E = 255$  e  $M \neq 0$  (NB: molte possibili rappresentazioni)
- $+\infty$ :  $S = 0$ ,  $E = 255; (11111111)_2$ ,  
 $M = 0$ ;  $(00000000000000000000000000000000)_2$
- $-\infty$ :  $S = 1$ ,  $E = 255; (11111111)_2$ ,  
 $M = 0$ ;  $(00000000000000000000000000000000)_2$
- 0: segno qualsiasi,  $E = 0$ ,  $M = 0$   
( $\Rightarrow$  due rappresentazioni, come se ci fossero  $+0$  e  $-0$ )



## Casi Particolari

Eccezione dei numeri con  $E = 0$  e  $M \neq 0$  (numeri che tendono ad essere vicini al limite della rappresentabilità)

- In questi casi il numero è in forma **denormalizzata**



## Standard IEEE 754-1985

- Tre diversi formati, differiscono nel numero totale dei bit utilizzati. Quelli più diffusi:
  - precisione singola: 32 bit
    - 1 bit segno, 8 bit esponente, 23 bit mantissa
  - precisione doppia: 64 bit
    - 1 bit segno, 11 bit esponente, 52 bit mantissa
  - precisione estesa: 128 bit
    - 1 bit segno, 15 bit esponente, 112 bit mantissa



## ESERCIZIO

1. Si fornisca la codifica binaria  $CP_2$  del numero -221 utilizzando il minor numero di bit necessari per una corretta rappresentazione
2. Si fornisca la codifica binaria in virgola mobile secondo lo standard IEEE 754-1985 a precisione singola del numero -221.0625
3. Si dica, giustificando la risposta, se la rappresentazione fornita al punto 2 è esatta oppure comporta qualche approssimazione



## Punto 1, rappresentazione in $CP_2$ di -221

Su 8 bit codifico  $[-2^7, 2^7 - 1] = [-128, 127]$  ...non sufficienti

Su 9 bit codifico  $[-2^8, 2^8 - 1] = [-256, 255]$  ...sufficienti!!!



## Punto 1, rappresentazione in $CP_2$ di -221

Su 8 bit codifico  $[-2^7, 2^7 - 1] = [-128, 127]$  ...non sufficienti

Su 9 bit codifico  $[-2^8, 2^8 - 1] = [-256, 255]$  ...sufficienti!!!

221	1
110	0
55	1
27	1
13	1
6	0
3	1
1	1
0	

221 = 11011101



## Punto 1, rappresentazione in $CP_2$ di -221

Su 8 bit codifico  $[-2^7, 2^7 - 1] = [-128, 127]$  ...non sufficienti

Su 9 bit codifico  $[-2^8, 2^8 - 1] = [-256, 255]$  ...sufficienti!!!

221	1	
110	0	$221 = 11011101$
55	1	011011101 (estendo a 9 bit)
27	1	100100010 (complemento)
13	1	100100011 (sommo 1)
6	0	
3	1	
1	1	
0		





## Punto 2. IEEE 754-1985 del numero -221.0625

- Parte intera: 221 in binario naturale 11011101



## Punto 2. IEEE 754-1985 del numero -221.0625

- Parte intera: 221 in binario naturale 11011101

0,0625 x 2

0,1250 x 2      0

0,250    x 2      0

0,50      x 2      0

1,0                      1

=> 0,0001

Nessuna  
approssimazione



## Punto 2. IEEE 754-1985 del numero -221.0625

- Parte intera: 221 in binario naturale 11011101

0,0625 x 2

0,1250 x 2      0

0,250    x 2      0

0,50      x 2      0

1,0                      1

=> 0,0001

• 221 = 11011101

• 0,0625 = 0,0001

221,0625 = 11011101,0001

Nessuna  
approssimazione



## Punto 2. IEEE 754-1985 del numero -221.0625

- Parte intera: 221 in binario naturale 11011101

0,0625 x 2

0,1250 x 2      0

0,250    x 2      0

0,50      x 2      0

1,0                      1

=> 0,0001

Nessuna  
approssimazione

• 221 = 11011101

• 0,0625 = 0,0001

221,0625 = 11011101,0001 =

= 1,10111010001 x 2<sup>7</sup> (normalizzo)



## Punto 2. IEEE 754-1985 del numero -221.0625

- Parte intera: 221 in binario naturale 11011101

$$0,0625 \times 2$$

$$0,1250 \times 2 \quad 0$$

$$0,250 \times 2 \quad 0$$

$$0,50 \times 2 \quad 0$$

$$1,0 \quad 1$$

$$\Rightarrow 0,0001$$

Nessuna  
approssimazione

- $221 = 11011101$

- $0,0625 = 0,0001$

$$221,0625 = 11011101,0001 =$$

$$= 1,10111010001 \times 2^7 \text{ (normalizzo)}$$

$$S = 1 \text{ (1 bit)}$$

$$E = 7 + 127 = 134 = 10000110 \text{ (8 bit)}$$

$$M = (1,)10111010001(+ 12 \text{ zeri}) \text{ (23 bit)}$$

$$\text{Quindi } 11000011010111010001(+ 12 \text{ zeri})(32 \text{ bit})$$

