



# La Rappresentazione dell'Informazione

Informatica, AA 2020/2021

Stefano Marelli

11/12/2020

[stefano1.marelli@polimi.it](mailto:stefano1.marelli@polimi.it)



## 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

- $3401 = 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$

- 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 10, il dizionario è  $A_{10} = \{0, \dots, 9\}$
- Un numero è una sequenza di  $m$  cifre
$$a_{m-1} \dots a_1 a_0 \text{ con } a_i \in A$$
  - 8522 è una sequenza di 4 cifre di  $A_{10}$ ,  $\{8,5,2\} \subset A_{10}$ .
- 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:

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

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

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

- 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  $[0, 9]$  i.e.,  $A_{10}$ )
  - $m = 3$  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  $A_{10}$ ,  $A_p = \{0, \dots, p - 1\}$
  - 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  $[0,28560]$ )

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



## Le targhe

Quante diverse targhe italiane ci possono essere (ignoriamo targhe speciali come CC , CRI, EI,..)



Lettere in uso: 24 (cioè 26 – 2, I e O non vengono utilizzate)

Nro di targhe :  $24^4 * 10^3 = 331776000$

Se usassi numeri in base 10: 9 cifre

Se usassi numeri in base 2  $\approx \log_2(331776000) = 23$  cifre



## 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





## Le potenze del 2

- È necessario imparare le potenze di 2!

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

- E i loro legami con l'informatica:
  - Byte = 8 bit
  - KiloByte (kB) =  $10^3$  Byte
  - MegaByte (MB) =  $10^6$  Byte
  - GigaByte (GB) =  $10^9$  Byte
  - TheraByte (TB) =  $10^{12}$  Byte



## Altre Codifiche che consideriamo

- Codifica **ottale** (in **base 8**)
  - $A_8 = \{0, 1, \dots, 7\}$
  - con  $m$  cifre in  $A_8$  scrivo i numeri da  $[0, 8^m - 1]$
  
- Codifica **esadecimale**, (in **base 16**)
  - $A_{16} = \{0, 1, \dots, 9, A, B, C, D, E, F\}$ ,
  - Per le conversioni  $A = 10, \dots, F = 15$ .
  - con  $m$  cifre in  $A_{16}$  scrivo i numeri da  $[0, 16^m - 1]$ .



## 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)



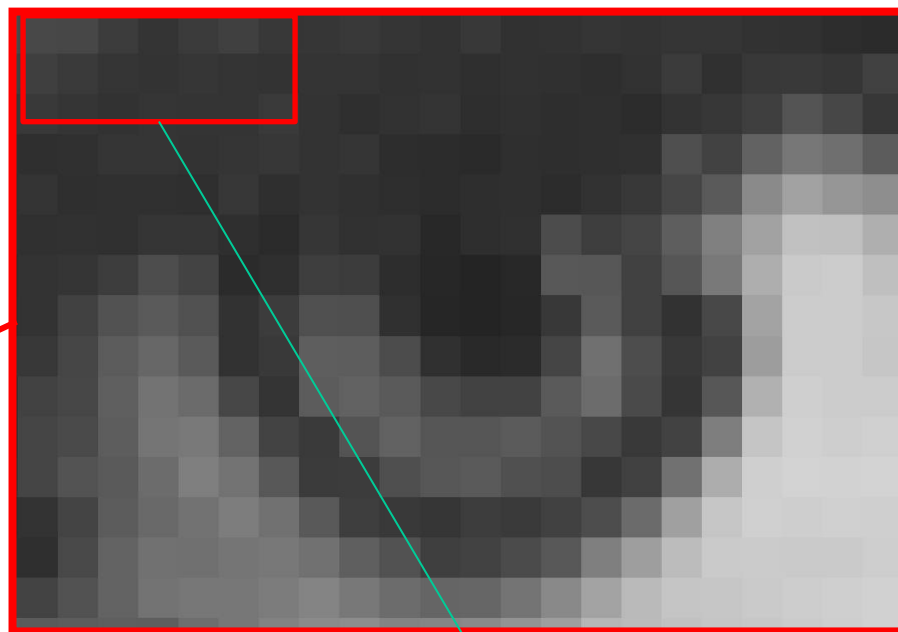
## La codifica ASCII (parziale)

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



## Codifica delle Immagini

- Esistono diverse codifiche dell'immagine, non sempre questa viene scritta pixel per pixel.
- È spesso conveniente rappresentare l'immagine secondo codifiche che permettano di ridurre le dimensioni.
- Codifiche *lossless*: permettono, senza perdita di informazione, di comprimere l'immagine
  - e.g, non serve ripetere il valore nelle aree costanti, conviene registrare le variazioni (e.g. gif, png)
- Codifiche *lossy*: comportano una perdita di informazione e di qualità
  - e.g., le immagini jpeg sono compresse a blocchi, ogni blocco contiene meno dettagli dell'immagine originale.



# Convertire in base 10

da base 2 (o altre basi) a base 10





## 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.*

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

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



## Osservazioni

- In binario i numeri che terminano con 1 sono dispari, quelli con 0 sono pari.
  - L'unico modo per avere un numero dispari nella somma è aggiungere  $2^0 = 1$
- Le conversioni di numeri con bit tutti a 1 si calcolano facilmente

$$(111111)_2 = (1000000)_2 - (1)_2 = 2^6 - 1 =$$



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



# Convertire in base 2

da base 10 (o altre basi) a base 2





## Conversione Decimale → Binario

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

- $531 / 2 = 265 + 1$

- $265 / 2 = 132 + 1$

- $132 / 2 = 66 + 0$

- $66 / 2 = 33 + 0$

- $33 / 2 = 16 + 1$

- $16 / 2 = 8 + 0$

- $8 / 2 = 4 + 0$

- $4 / 2 = 2 + 0$

- $2 / 2 = 1 + 0$

- $1 / 2 = 0 + 1$

Divisione intera tra il numero e 2

Il risultato della divisione precedente viene successivamente diviso

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:

•  $531 / 2 = 265 + 1$   Cifra meno significativa

•  $265 / 2 = 132 + 1$

•  $132 / 2 = 66 + 0$

•  $66 / 2 = 33 + 0$


•  $33 / 2 = 16 + 1$

•  $16 / 2 = 8 + 0$

•  $8 / 2 = 4 + 0$

•  $4 / 2 = 2 + 0$

•  $2 / 2 = 1 + 0$

•  $1 / 2 = 0 + 1$   Cifra più significativa

I resti della divisione intera, letti dall'ultimo al primo, identificano il numero binario

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



## Algoritmo per convertire da base 10 a base 2

Assumiamo di scrivere il numero binario in un'opportuna struttura dati: un array!

1. Sia  $n$  il numero da convertire da base 10 a base 2
2. Se  $n = 0$  oppure  $n = 1$ , allora  $n$  è già in base 2

Altrimenti

3. Salva il resto della divisione tra  $n$  e 2 in una cella di un array
4. Associa ad  $n$  la divisione intera tra  $n$  e 2
5. Ripeti 3 – 4 fino a quando  $n == 0$
6. Leggi l'array al contrario per ottenere  $n$  in base 2



## Conversioni ottale/esadecimale $\leftrightarrow$ decimale

Convertire in base 2 i seguenti numeri

- $(31)_8 = (25)_{10}$
- $(A170)_{16} = (41328)_{10}$
- $(623)_8 = (403)_{10}$
- $(623)_{16} = (1571)_{10}$



## Conversioni ottale/esadecimale $\rightarrow$ decimale

Convertire in base 2 i seguenti numeri

- $(31)_8 = (25)_{10} = (11001)_2$
- $(A170)_{16} = (41328)_{10} = TBD$
- $(623)_8 = (403)_{10} = (110010011)_2$
- $(623)_{16} = (1571)_{10} = (11000100011)_2$



## Conversioni ottale/esadecimale → binario

- È possibile passare in base 10 e quindi utilizzare l'algoritmo delle divisioni successive
- È tuttavia più comodo fare delle conversioni direttamente dalla rappresentazione binaria e
- Esprimere ogni sequenza di 3 numeri binari in base 8
  - $(1231)_{10} = (10011001111)_2 = (\underbrace{010}_{2} \underbrace{011}_{3} \underbrace{001}_{1} \underbrace{111}_{7})_2$
  - $(1231)_{10} = (2317)_8$
- Esprimere ogni sequenza di 4 numeri binari in base 16
  - $(1231)_{10} = (10011001111)_2 = (\underbrace{0100}_{4} \underbrace{1100}_{C} \underbrace{1111}_{F})_2$
  - $(1231)_{10} = (4CF)_{16}$
- Queste tecniche possono essere usate per convertire da base decimale in esadecimale/ottale passando facilmente in binario



## Conversioni ottale/esadecimale $\leftrightarrow$ decimale

Convertire in base 2 i seguenti numeri

- $(31)_8 = (25)_{10} = (011\ 001)_2$
- $(A170)_{16} = (41328)_{10} = (1010\ 0001\ 0111\ 000)_2$
- $(623)_8 = (403)_{10} = (110\ 010\ 011)_2$
- $(623)_{16} = (1571)_{10} = (0110\ 0010\ 0011)_2$

Ora è più facile convertire *A170*



# Somma tra Numeri Interi Positivi

Somma in base 2





## 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

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

$$\begin{array}{r} \mathbf{111} \xrightarrow{\text{Riporto}} \\ \mathbf{1111} + (15)_{10} \\ \mathbf{1010} = (10)_{10} \\ \hline \mathbf{(1)1001} \quad (25)_{10} \end{array}$$



## 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**

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

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



# I 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
- Con  $m$  cifre in binario e codifica modulo dedico  $2^{m-1}$  per i positivi e  $2^{m-1}$  per gli stessi cambiati di segno
  - posso rappresentare tutti i numeri nell'intervallo

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

- Es
  - 01010 = + 10
  - 11101 = - 13
  - -27 = 111011
  - 122 = 01111010



## Rappresentazione Modulo e Segno

- Esempio  $m = 3$

- $0 = 000$

- $1 = 001$

- $2 = 010$

- $3 = 011$

- $-0 = 100$

- $-1 = 101$

- $-2 = 110$

- $-3 = 111$

Ho due codifiche differenti lo zero

- C'è uno «spreco» nella codifica
- Ostacola realizzazione circuitale delle operazioni algebriche (non lo mostriamo)
- Occorre trovare una rappresentazione migliore!



## 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**
- Questo equivale alla seguente codifica:

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



## 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**
- Questo equivale alla seguente codifica:

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

i.e., viene cambiato il segno dell'addendo relativo alla cifra più significativa





## Rappresentazione in CP2

- Esempio  $m = 3$ 
  - $-4 =$
  - $-3 =$
  - $-2 =$
  - $-1 =$
  - $0 =$
  - $1 =$
  - $2 =$
  - $3 =$



## Rappresentazione in CP2

- Esempio  $m = 3$ 
  - $-4 = 100$
  - $-3 = 101$
  - $-2 = 110$
  - $-1 = 111$
  - $0 = 000$
  - $1 = 001$
  - $2 = 010$
  - $3 = 011$



## Rappresentazione in CP2

- Con i positivi copro solo il range  $[0, 2^{m-1}-1]$ , quindi la prima cifra è 0 (il numero è minore di  $2^{m-1}$ )
- Con i negativi copro il range  $[-2^{m-1}, -1]$  e scrivo  $2^m - |X|$ , e quindi la prima cifra è 1 (il numero è maggiore di  $2^{m-1}$ )
- Quindi, il primo bit **indica il segno** del numero
  - Attenzione: questo numero **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 (cioè  $m$  volte 0), nessuna configurazione “sprecata” dalla codifica



## 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  $[-2^5, 2^5 - 1] = [-32, 32]$  (non cont. 45)~~
  - $-23 \rightarrow 2^7 - 23 = 128 - 23 = 105 = (1101001)_{CP2}$
  - $45 = (0101101)_{CP2}$
- **NB:** occorre utilizzare **sempre  $m$  bit**. Se non avessi messo lo 0 iniziale in  $(45)_{CP2}$  avrei ottenuto un numero negativo a 6 bit!



Metodo "operativo" per rappresentare  $X$  ad  $m$  bit

1. Controllo che  $X \in [-2^{m-1}, 2^{m-1} - 1]$ , altrimenti  $m$  bit non bastano
2. Se  $X$  è positivo, scrivo  $X$  utilizzando  $m$  bit  
**NB:** ricordandosi di aggiungerei zeri se necessario all'inizio del numero!
3. Se  $X$  è negativo:
  - a) Scrivo  $|X|$  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  $[-2^6, 2^6 - 1] = [-64, 63]$

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

iii. Complemento  $\rightarrow 1000111$

iv. Sommo  $1 \quad \underline{\quad \quad \quad 1}$

v.  $(1001000)_{CP2} = (-56)_{10}$

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



Esercizio: convertire in complemento a 2 i seguenti numeri, utilizzando il numero di bit necessario per esprimerli tutti

$$(12)_{10} =$$

$$(-12)_{10} =$$

$$(-8)_{10} =$$

$$(1)_{10} =$$

$$(-101)_{10} =$$

$$(-54)_{10} =$$





Esercizio: convertire in complemento a 2 i seguenti numeri, utilizzando il numero di bit necessario per esprimerli tutti

$$(12)_{10} = (0000\ 1100)_{CP2}$$

$$(-12)_{10} = (1111\ 0100)_{CP2}$$

$$(-8)_{10} = (1111\ 1000)_{CP2}$$

$$(1)_{10} = (0000\ 0001)_{CP2}$$

$$(-101)_{10} = (1001\ 1011)_{CP2}$$

$$(-54)_{10} = (1100\ 1010)_{CP2}$$



Possiamo utilizzare la definizione

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

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

$$\begin{aligned} (10011011)_{CP2} &= -2^7 + 2^4 + 2^3 + 2^1 + 2^0 = \\ &= -128 + 16 + 8 + 2 + 1 = (-101)_{10} \end{aligned}$$

**NB** convertite sempre in decimale con questo metodo per controllare le vostre operazioni



## Conversione CP2 ➔ Decimale

... in alternativa è possibile utilizzare un metodo operativo:

1. Se  $(X)_{CP2}$  inizia per 0, allora è positivo: lo converto normalmente
2. Se  $(X)_{CP2}$  inizia per 1, allora è negativo
  - a) **Complemento** tutti i bit di  $(X)_{CP2}$  ( $1 \rightarrow 0, 0 \rightarrow 1$ )
  - b) **Sommo 1** al numero ottenuto
  - c) **Converto** in decimale e cambio di segno



## Esempio

Esercizio: riconvertire in decimale i seguenti numeri in complemento a 2

$$\left\{ \begin{array}{l} (1001010)_{CP2} \rightarrow (0110101) \rightarrow (0110110) \rightarrow (-54)_{10} \\ (1001010)_{CP2} = -2^6 + 2^3 + 2^1 = -64 + 8 + 2 = -54 \end{array} \right.$$

$$(011)_{CP2}$$

$$(1101001)_{CP2}$$

$$(11111)_{CP2}$$

$$(10100)_{CP2}$$

$$(101)_{CP2}$$



## 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 **ignorare il bit di carry**, cioè il riporto che cade sul bit che cade sul segno
- 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





## Esempio

Esempio:  $(100)_{CP2} + (101)_{CP2}$

1

1 0 0 +

1 0 1 =

---

[1] (1) 0 0 1

- Ignoro il bit di carry
- **Overflow:** la somma di due numeri negativi mi ha dato un numero positivo.
- L'overflow si indica quadre: [1] c'è overflow, [0] non c'è
- 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
- Indico tra () bit di carry, tra [] bit di overflow

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

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

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

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

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



## Esempi

- Esempi: con  $m = 4$  bit
- Indico tra () bit di carry, tra [] bit di overflow

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad \underline{1100} \\ -7 \Rightarrow [0] (1)1001 \end{array}$$

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

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

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

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



## Esempi

- Esempi: con  $m = 4$  bit
- Indico tra () bit di carry, tra [] bit di overflow

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad \underline{1100} \\ -7 \Rightarrow [0] (1)1001 \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ +6 \Rightarrow \quad \underline{0110} \\ +3 \Rightarrow [0] (1)0011 \end{array}$$

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

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

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



## Esempi

- Esempi: con  $m = 4$  bit
- Indico tra () bit di carry, tra [] bit di overflow

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad \underline{1100} \\ -7 \Rightarrow [0] (1)1001 \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ +6 \Rightarrow \quad \underline{0110} \\ +3 \Rightarrow [0] (1)0011 \end{array}$$

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

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

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



## Esempi

- Esempi: con  $m = 4$  bit
- Indico tra () bit di carry, tra [] bit di overflow

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad \underline{1100} \\ -7 \Rightarrow [0] (1)1001 \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ +6 \Rightarrow \quad \underline{0110} \\ +3 \Rightarrow [0] (1)0011 \end{array}$$

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

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

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



## Esempi

- Esempi: con  $m = 4$  bit
- Indico tra () bit di carry, tra [] bit di overflow

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ -4 \Rightarrow \quad \underline{1100} \\ -7 \Rightarrow [0] (1)1001 \end{array}$$

$$\begin{array}{r} -3 \Rightarrow \quad 1101 \\ +6 \Rightarrow \quad \underline{0110} \\ +3 \Rightarrow [0] (1)0011 \end{array}$$

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

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

$$\begin{array}{r} +3 \Rightarrow \quad 0011 \\ +6 \Rightarrow \quad \underline{0110} \\ +9 \Rightarrow [1](0) 1001 \end{array}$$



## Esempio TDE 11/2009

- a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.
- b) Con riferimento a tale codifica indicare, giustificando brevemente le risposte, quali delle seguenti operazioni possono essere effettuate correttamente:
- i.  $-254 - 255$
  - ii.  $+254 - 253$
  - iii.  $-18 + 236$
  - iv.  $+217 + 182$
- 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 TDE 11/2009

- a. Valori rappresentabili vanno da -256 a +255.
- b. Le soluzioni:
  - i. -254 – 255: NO, si ottiene un valore negativo troppo grande in valore assoluto
  - ii. + 254 – 253: SI, si ottiene un valore piccolo in valore assoluto
  - iii. -18 + 236: SI, si ottiene un valore positivo, grande in valore assoluto ma nei limiti
  - iv. + 217 + 182: NO, si ottiene un valore positivo troppo grande in valore assoluto

c.	100000010 (-254)	ii. 011111110 (+254)
	<u>100000001 (-255)</u>	<u>100000011 (-253)</u>
	[1](1)000000011 (-509)	[0](1)000000001 (+1)





Convertire il messaggio pubblicitario in base 10 assumendo sia scritto in CP2

MediaOne

PER TUTTI I CLIENTI  
ENEL ENERGIA LUCE

**OFFERTA GAS**

**-20%**

PER 12 MESI HAI  
IL 20% DI SCONTO  
SULLA COMPONENTE MATERIA PRIMA GAS

CHIAMA ENEL ENERGIA  
**800 900 860**

PER TUTTI GLI ALTRI  
UN CODICE BINARIO

01100101 01101110  
01100101 01110010  
01100111 01101001  
01100001 00001101

enel.it

INFORMATICA AA 19/20, Giacomo Boracchi