

Laboratorio I

a.a. 2026/2027

Rappresentazione binaria

Contenuti

- Rappresentazione posizionale in base B
- Rappresentazione binaria degli interi
 - Segno e modulo
 - Complemento a 2
- Rappresentazione binaria dei numeri reali
- Altri tipi di dato
- Operazioni binarie

Rappresentazione posizionale in base 10

$$7452 = 7*1000 + 4*100 + 5*10 + 2*1 = 7*10^3 + 4*10^2 + 5*10^1 + 2*10^0$$

posizione

cifre

$$\text{cifre} = \{0, 1, \dots, 9\} = \{0, 1, \dots, B-1\}$$

$$B = \text{base} = 10$$

Numeri naturali in base 10:

$$\sum_{i=0}^{N-1} n_{10} = d_i * 10^i$$

$d_0 \rightarrow$ cifra meno significativa

$d_{N-1} \rightarrow$ cifra più significativa

Rappresentazione posizionale in base 2 - binaria

$$\text{cifre} = \{0, 1\} = \{0, 2-1\}$$

$$B = \text{base} = 2$$

Numeri naturali in base 2:

$$\sum_{i=0}^{N-1} n_2 = b_i * 2^i$$

b_0 → bit meno significativo (Least Significant Bit, LSb)

b_{N-1} → bit più significativo (Most Significant Bit, MSb)

MSb ↗ ↘ LSb

$$10011010_2 = 1*128 + 0*64 + 0*32 + 1*16 + 1*8 + 0*4 + 1*2 + 0*1 =$$
$$1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 154_{10}$$

Massimo naturale su N bit:

$$2^N - 1 = \sum_{i=0}^{N-1} 2^i$$

Da base 10 a base 2

154		

Esercizio:

Funzione che prende un numero naturale in base 10 e restituisce le cifre in base 2:

$$154_{10} = 10011010_2$$

Pensiamoci prima in base 10:

Come **estrarre** le cifre di
 $154_{10} = 1 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$?

Da base 10 a base 2

154	quoziente	resto
/ 2	77	0

Esercizio:

Funzione che prende un numero naturale in base 10 e restituisce le cifre in base 2:

$$154_{10} = 10011010_2$$

Pensiamoci prima in base 10:

Come **estrarre** le cifre di $154_{10} = 1 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$?

Da base 10 a base 2

154	quoziente	resto
/ 2	77	0
/ 2	38	1

Esercizio:

Funzione che prende un numero naturale in base 10 e restituisce le cifre in base 2:

$$154_{10} = 10011010_2$$

Pensiamoci prima in base 10:

Come **estrarre** le cifre di
 $154_{10} = 1 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$?

Da base 10 a base 2

154	quoziente	resto
/ 2	77	0
/ 2	38	1
/ 2	19	0
/ 2	9	1
/ 2	4	1
/ 2	2	0
/ 2	1	0
/ 2	0	1

Esercizio:

Funzione che prende un numero naturale in base 10 e restituisce le cifre in base 2:

$$154_{10} = 10011010_2$$

Pensiamoci prima in base 10:

Come **estrarre** le cifre di
 $154_{10} = 1 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$?

Base generica

Numeri naturali in base **B**:

$$n_B = \sum_{i=0}^{N-1} a_i \cdot B^i$$

Esempi:

❖ **cifre** = $\{0, \dots, 7\} = \{0, \dots, 8-1\}$ **B** = base = **8**

$$10000047_8 = 1 \cdot 2097152 + 4 \cdot 8 + 7 \cdot 1 = 1 \cdot 8^7 + 4 \cdot 8^1 + 7 \cdot 8^0 = \mathbf{2097191}_{10}$$

❖ **cifre** = $\{0, \dots, 9, A, \dots, F\} = \{0, \dots, 16-1\}$ **B** = base = **16**

$$100C0A47_{16} = 1 \cdot 16^7 + 12 \cdot 16^4 + 10 \cdot 16^2 + 4 \cdot 16^1 + 7 \cdot 16^0 = \mathbf{269224519}_{10}$$

Base 8 e 16

La rappresentazione **ottale** (B=8) ed **esadecimale** (B=16) sono utili per scrivere i numeri binari in forma compatta:

Considero gruppi
di 3 bit da dx a sx

$$\begin{array}{ccccccc} & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ & \underbrace{} & \underbrace{} & \underbrace{} & & & & & \\ (2 & 3 & 2) & \end{array} = 154_{10}$$
$$(2 \quad 3 \quad 2)_8 = 154_{10}$$

Considero gruppi
di 4 bit da dx a sx

$$\begin{array}{ccccccc} & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ & \underbrace{} & \underbrace{} & & & & & & \\ (9 & A) & \end{array} = 154_{10}$$
$$(9 \quad A)_{16} = 154_{10}$$

Rappresentazione dell'informazione

- I computer usano la rappresentazione binaria
- Circuiti elettronici la possono rappresentare e fare operazioni facilmente (e a costo basso) **ON/OFF**, porte logiche
 - livelli di tensione, tradizionalmente 0v e +5v, anche se oggi non sono esattamente questi i valori

Bit: unità elementare di informazione, ha (solo) due stati — il minimo perché si possa parlare di cose diverse. Convenzionalmente, li denotiamo con 0 e 1

- *Due stati più semplici da distinguere* piuttosto che 8 (se consideriamo che stiamo parlando di valori di corrente, che non sempre possono essere “puliti”)

Byte: 8 bit, unità elementare di memoria, 2^8 valori distinti

Parola di memoria: 8, 16, 32, 64, ... bit dimensione dei registri nella CPU.

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = ?$$

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

$$\begin{array}{r} 10011010 \quad + \\ \hline 11111111 \quad = \end{array}$$

Proviamo a fare la somma

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

riporti:

$$\begin{array}{r} 10011010 \quad + \\ 11111111 \quad = \\ \hline 1 \end{array}$$

Proviamo a fare la somma

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

riporti:

$$\begin{array}{r} 1 \\ 10011010 + \\ 11111111 = \\ \hline 01 \end{array}$$

Proviamo a fare la somma

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

riporti:

$$\begin{array}{r} 11 \\ 10011010 + \\ 11111111 = \\ \hline 001 \end{array}$$

Proviamo a fare la somma

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

riporti: **1**111111

$$\begin{array}{r} 10011010 \\ + 11111111 \\ \hline \end{array}$$

110011001

Rappresentazione dell'informazione - **overflow**

$$10011010_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 154_{10}$$

$$11111111_2 = 255_{10}$$

riporti: **1**1111111

$$\begin{array}{r} 10011010 \\ + 11111111 \\ \hline \end{array}$$

110011001



Si ha **overflow** quando c'è un riporto sul MSb

Non riesco a rappresentare il risultato dell'operazione con lo stesso numero di bit degli operandi!

Numeri interi: **rappresentazione** con modulo e segno

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in [????, ?????]

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in [????, ?????]

8 bit:

+-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Segno	Modulo							Base 10
0	0	1	1	0	0	1	0	+50
1	0	0	1	0	0	0	1	-17

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

8 bit:

+-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Segno	Modulo							Base 10
0	0	1	1	0	0	1	0	+50
1	0	0	1	0	0	0	1	-17

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

8 bit:

+-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Segno	Modulo							Base 10
0	0	1	1	0	0	1	0	+50
1	0	0	1	0	0	0	1	-17

Problemi:

- In quanti modi posso rappresentare il numero 0?**

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

8 bit:

+-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Segno	Modulo							Base 10
0	0	1	1	0	0	1	0	+50
1	0	0	1	0	0	0	1	-17

Problemi:

- 2 rappresentazioni diverse per zero: 0000 0000 e 1000 0000 (+0 e -0)
- **Cosa potrebbe succedere se sommo due positivi?**

Numeri interi: **rappresentazione** con modulo e segno

Primo bit (**bit più significativo**, most significant bit, MSb) usato per il segno

+ → MSb=0

- → MSb=1

Altri bit usati per il modulo → su N bit rappresentiamo interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

8 bit:

+-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Segno	Modulo							Base 10
0	0	1	1	0	0	1	0	+50
1	0	0	1	0	0	0	1	-17

Problemi:

- 2 rappresentazioni diverse per zero: 0000 0000 e 1000 0000 (+0 e -0)
- Somma di 2 positivi è negativa se riporto sul bit di segno

Numeri interi: **rappresentazione** con modulo e segno

Nella rappresentazione in **Modulo e Segno** con n bit
quanti numeri posso
rappresentare?

S	M	
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 0
1	01	- 1
1	10	- 2
1	11	- 3

su N bit rappresentiamo
interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

Problemi:

- 2 rappresentazioni diverse per zero: 0000 0000 e 1000 0000 (+0 e -0)
- Somma di 2 positivi è negativa se riporto sul bit di segno

Numeri interi: **rappresentazione** con modulo e segno

Nella rappresentazione in **Modulo e Segno** con n bit quanti numeri posso rappresentare?

$$2^n - 1$$

S	M	
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 0
1	01	- 1
1	10	- 2
1	11	- 3

su N bit rappresentiamo interi in $[-2^{(N-1)}+1, 2^{(N-1)}-1]$

Problemi:

- 2 rappresentazioni diverse per zero: 0000 0000 e 1000 0000 (+0 e -0)
- Somma di 2 positivi è negativa se riporto sul bit di segno

Numeri interi: **rappresentazione** con modulo e segno

Provate....

Segno	Modulo							Base 10
0	0	0	0	0	0	0	1	+1
1	0	1	0	0	1	0	1	-37

Numeri interi: **rappresentazione** con modulo e segno

Provate....

Segno	Modulo							Base 10
0	0	0	0	0	0	0	1	+1
1	0	1	0	0	1	0	1	-37
1	0	1	0	0	1	1	0	-38

Problemi:

- L'operazione di somma fornisce risultati sbagliati quando gli addendi hanno segni diversi

Numeri interi: **rappresentazione** con modulo e segno

Soluzione: Quando si esegue una somma si dovrebbero verificare i segni dei due numeri e, se risultano diversi, si dovrebbe eseguire una sottrazione. *Ciò costringe il calcolatore ad eseguire un'operazione di verifica in più prima di eseguire l'addizione*

Problemi:

- L'operazione di somma fornisce risultati sbagliati quando gli addendi hanno segni diversi

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

179

$(256 - 1) - 179 = 76$

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

179

$(256 - 1) - 179 = 76$

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

179

$(256 - 1) - 179 = 76$

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Esempio (su 4 bit):

- 3_{10} : ??
- -3_{10} : ??

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

179

$(256 - 1) - 179 = 76$

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Esempio (su 4 bit):

- 3_{10} : 0011_2
- -3_{10} : ??

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N-1) - a$

179

$(256-1) - 179 = 76$

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Esempio (su 4 bit):

- $3_{10}: 0011_2$
- $-3_{10}: 0011_2 = 3_{10} \rightarrow 1100_2$

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

179

$(256 - 1) - 179 = 76$

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Esempio (su 4 bit):

- 3_{10} : 0011_2
- -3_{10} : $0011_2 = 3_{10} \rightarrow 1100_2$

Nella rappresentazione **complemento a uno** con n bit **posso rappresentare i numeri in quale intervallo?**

Numeri interi: complemento a 1

000	????
001	????
010	????
011	????
100	????
101	????
110	????
111	????

Nella rappresentazione **complemento a uno** con n bit **posso rappresentare i numeri in quale intervallo?**

Numeri interi: complemento a 1

000	+0
001	1
010	2
011	3
100	-3
101	-2
110	-1
111	-0

Nella rappresentazione **complemento a uno** con n bit **posso rappresentare i numeri in quale intervallo?**

Numeri interi: complemento a 1

Invertiamo tutti i bit: $a=10110011 \rightarrow \bar{a}=01001100$

Matematicamente: $\bar{a} = (2^N - 1) - a$

**Soliti problemi di
modulo e segno**

Nel metodo di rappresentazione complemento a 1:

- Se il numero da codificare è **positivo** lo si **converte** in binario
- Se il numero è **negativo** si **converte** in binario il suo valore assoluto e si esegue l'operazione di **complementazione**

Esempio (su 4 bit):

- 3_{10} : 0011_2
- -3_{10} : $0011_2 = 3_{10} \rightarrow 1100_2$

Nella rappresentazione **complemento a uno** con n bit **posso rappresentare i numeri in quale intervallo?**

$$[-(2^{n-1} - 1), 2^{n-1} - 1]$$

Numeri interi: complemento a 2

Rispetto al modulo e segno **cambia il peso** assegnato al MSb: **il MSb pesa -2^{n-1}**

Su $n=8$ bit, il peso del MSb è $-2^{n-1} = -128$

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	0	0	1	0	1	
$1*(-2^7)$	$0*2^6$	$1*2^5$	$0*2^4$	$0*2^3$	$1*2^2$	$0*2^1$	$1*2^0$	-91

Numeri interi: complemento a 2

Rispetto al modulo e segno **cambia il peso** assegnato al MSb: **il MSb pesa -2^{n-1}**

Su $n=8$ bit, il peso del MSb è $-2^{n-1} = -128$

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	0	0	1	0	1	
$1*(-2^7)$	$0*2^6$	$1*2^5$	$0*2^4$	$0*2^3$	$1*2^2$	$0*2^1$	$1*2^0$	-91

- Massimo numero (positivo) rappresentabile?
- Minimo numero (negativo) rappresentabile?

Numeri interi: complemento a 2

Rispetto al modulo e segno **cambia il peso** assegnato al MSb: il **MSb pesa -2^{n-1}**

Su $n=8$ bit, il peso del MSb è $-2^{n-1} = -128$

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	0	0	1	0	1	
$1*(-2^7)$	$0*2^6$	$1*2^5$	$0*2^4$	$0*2^3$	$1*2^2$	$0*2^1$	$1*2^0$	-91

- Massimo numero (positivo) rappresentabile? **127 ($2^{n-1} - 1$):** 0111 1111
- Minimo numero (negativo) rappresentabile? **-128 (-2^{n-1}):** 1000 0000

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Esempio: esprimere $k=-5$ su 8 bit

- Numero che sommato a -128 dà -5? ????
- Rappresentazione di 123 in base 2: ????
- Avendo il MSb a 1, abbiamo: 1 ????

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Esempio: esprimere $k=-5$ su 8 bit

- Numero che sommato a -128 dà -5? **123**

- Rappresentazione di 123 in base 2: ????

- Avendo il MSb a 1, abbiamo: 1 ????

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Esempio: esprimere $k=-5$ su 8 bit

- Numero che sommato a -128 dà -5? **123**
- Rappresentazione di 123 in base 2: **1111011**
- Avendo il MSb a 1, abbiamo: 1 ????

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Esempio: esprimere $k=-5$ su 8 bit

- Numero che sommato a -128 dà -5? **123**
- Rappresentazione di 123 in base 2: **1111011**
- Avendo il MSb a 1, abbiamo: 1 **1111011**

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

→ Quanto vale in complemento a 2 il numero 1111 1111?

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

→ Quanto vale in complemento a 2 il numero 1111 1111?

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	
$1*(-2^7)$	$1*2^6$	$1*2^5$	$1*2^4$	$1*2^3$	$1*2^2$	$1*2^1$	$1*2^0$	-1

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

→ Quanto vale in complemento a 2 il numero 1111 1111? **Vale -1**

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	
$1*(-2^7)$	$1*2^6$	$1*2^5$	$1*2^4$	$1*2^3$	$1*2^2$	$1*2^1$	$1*2^0$	-1

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

- Quanto vale in complemento a 2 il numero 1111 1111? **Vale -1**
- Dato un numero a , quanto fa $a + \bar{a}$?

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

→ Quanto vale in complemento a 2 il numero 1111 1111? **Vale -1**

→ Dato un numero a , quanto fa $a + \bar{a}$? **1111 1111**

Numeri interi: complemento a 2

- Tutti i numeri con bit più significativo uguale a 1 sono negativi (come prima)
- Per stabilire la codifica di un generico numero negativo k , sapendo che necessariamente il bit più significativo va posto a 1, è sufficiente riportare nei restanti bit il numero positivo che, sommato a -2^{n-1} , dà il valore di k

Osservazione:

- Quanto vale in complemento a 2 il numero 1111 1111? **Vale -1**
- Dato un numero a , quanto fa $a + \bar{a}$? **1111 1111**
- Quindi, **in alternativa**, dato che $a + \bar{a} = -1$, **come possiamo ricavare $-a$?**

Numeri interi: complemento a 2

Oppure, **in alternativa**:

a'' : **inverto tutti i bit** di a poi sommo 1 $\rightarrow \bar{a}'' = a + 1$

- Rappresentazione dei numeri negativi: complemento a 2 del modulo (positivo)
- Rappresentazione dei numeri positivi: segno e modulo

Osservazione:

- Quanto vale in complemento a 2 il numero 1111 1111? **Vale -1**
- Dato un numero a , quanto fa $a + \bar{a}$? **1111 1111**
- Quindi, **in alternativa**, dato che $a + \bar{a} = -1$, **come possiamo ricavare $-a$?**

Numeri interi: complemento a 2

Esempio (su 8 bit):

$$a = 5_{10} = 0000\ 0101_2$$

$$a'' = 1111\ 1010 + 1 = 1111\ 1011 \rightarrow \text{rappresentazione di } (-5)$$

	BINARIO PURO	Comple- mento a 2
000 =	0	0
001 =	1	+ 1
010 =	2	+ 2
011 =	3	+ 3
100 =	4	- 4
101 =	5	- 3
110 =	6	- 2
111 =	7	- 1

Numeri interi: complemento a 2

Esempio (su 8 bit):

$$a = 5_{10} = 0000\ 0101_2$$

$$a'' = 1111\ 1010 + 1 = 1111\ 1011 \rightarrow \text{rappresentazione di } (-5)$$

	BINARIO PURO	Comple- mento a 2
000 =	0	0
001 =	1	+ 1
010 =	2	+ 2
011 =	3	+ 3
100 =	4	- 4
101 =	5	- 3
110 =	6	- 2
111 =	7	- 1

Vantaggi:

- Un solo 0 (0000 0000), se ne faccio complemento a 2 fa sempre 0000 0000
- L'operazione somma non ha bisogno di controllare il segno
- La sottrazione $a - b$ è la somma $a + b''$

Overflow rimane - determinato dalla dimensione della parola di memoria

Numeri interi: complemento a 2 - controllate da voi!

Esempi (parola di N=8 bit):

$$8+12 = 0000\ 1000_2 + 0000\ 1100_2 = 0001\ 0100_2 = 20_{10}$$

$$-4 - 2 = 1111\ 1100_2 + 1111\ 1110_2 = 1111\ 1010_2 = -6_{10}$$

$$8-16 = 0000\ 1000_2 + 1111\ 0000_2 = 1111\ 1000_2 = -8_{10}$$

$$100+100 = 0110\ 0100_2 + 0110\ 0100_2 = 1100\ 1000_2 = -56_{10} \quad \text{Overflow}$$

$$-100-100 = 1001\ 1100_2 + 1001\ 1100_2 = 0011\ 1000_2 = 56_{10} \quad \text{Overflow}$$

Overflow: quando *il risultato della somma di due numeri con lo stesso segno ha il segno opposto*

Numeri reali

$$\begin{array}{ccccccc} & & \text{segno} & & \text{esponente} & & \\ \pm 21.56 = \pm 0.2156 * 10^2 & & & & & & (\pm, 2156, 2) \\ \text{numero} & & \text{mantissa} & & & & \\ \text{reale} & & \text{sempre della forma 0.x} & & & & \end{array}$$

Numeri reali

$$\begin{array}{c} \text{segno} \\ \pm 21.56 = \pm 0.2156 * 10^2 \end{array} \quad (\pm, 2156, 2)$$

numero reale mantissa sempre della forma 0.x

IEEE 754

$$\pm r = \pm m * 2^E$$

SP (32 bit)

segno	esponente	mantissa
1	8	23

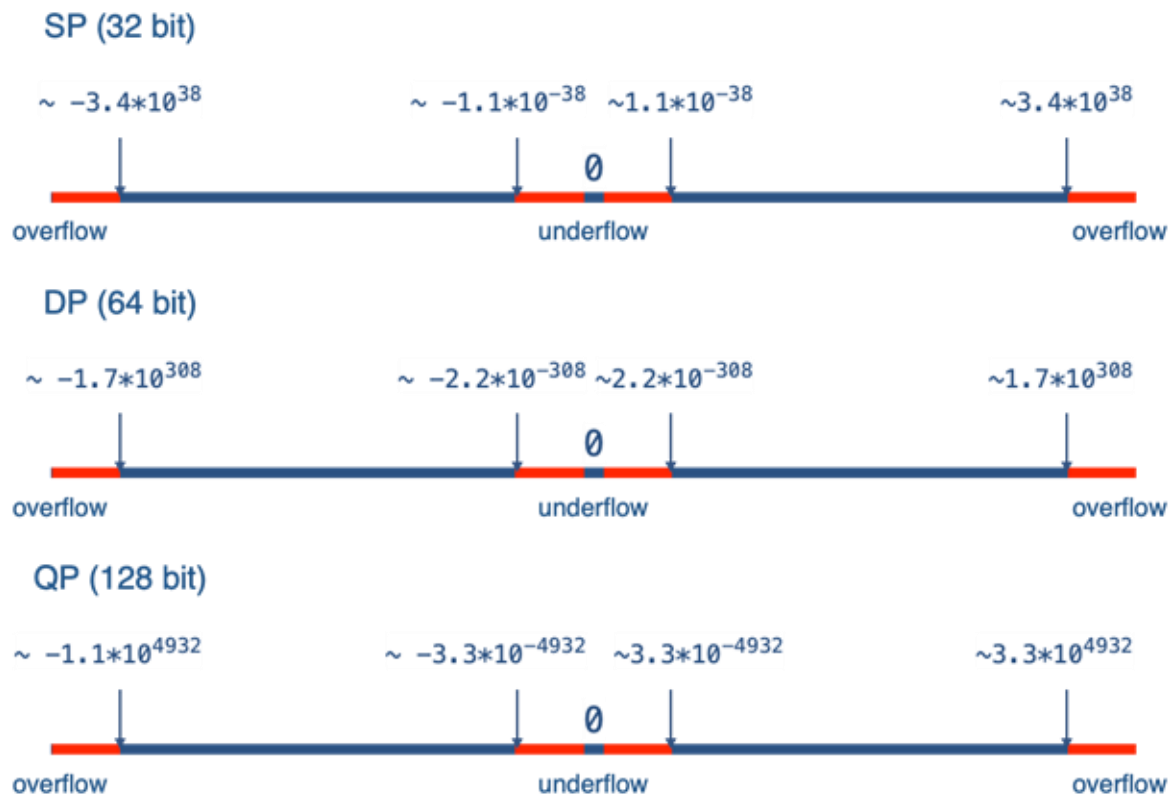
DP (64 bit)

segno	esponente	mantissa
1	11	52

QP (128 bit)

segno	esponente	mantissa
1	15	112

Numeri reali: intervalli di rappresentazione



Dati testuali

ASCII - American Standard
Code for Information
Interchange

dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
0	0	00	NUL (null)	32	40	20	(space)	64	100	40	@	96	140	60	`
1	1	01	SOH (start of header)	33	41	21	!	65	101	41	A	97	141	61	a
2	2	02	STX (start of text)	34	42	22	"	66	102	42	B	98	142	62	b
3	3	03	ETX (end of text)	35	43	23	#	67	103	43	C	99	143	63	c
4	4	04	EOT (end of transmission)	36	44	24	\$	68	104	44	D	100	144	64	d
5	5	05	ENQ (enquiry)	37	45	25	%	69	105	45	E	101	145	65	e
6	6	06	ACK (acknowledge)	38	46	26	&	70	106	46	F	102	146	66	f
7	7	07	BEL (bell)	39	47	27	'	71	107	47	G	103	147	67	g
8	10	08	BS (backspace)	40	50	28	(	72	110	48	H	104	150	68	h
9	11	09	HT (horizontal tab)	41	51	29	)	73	111	49	I	105	151	69	i
10	12	0a	LF (line feed - new line)	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	0b	VT (vertical tab)	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	0c	FF (form feed - new page)	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	0d	CR (carriage return)	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	0e	SO (shift out)	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	0f	SI (shift in)	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	DLE (data link escape)	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	DC1 (device control 1)	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	DC2 (device control 2)	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	DC3 (device control 3)	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	DC4 (device control 4)	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	NAK (negative acknowledge)	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	SYN (synchronous idle)	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	ETB (end of transmission block)	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CAN (cancel)	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	EM (end of medium)	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	SUB (substitute)	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	ESC (escape)	59	73	3b	;	91	133	5b	[	123	173	7b	{
28	34	1c	FS (file separator)	60	74	3c	<	92	134	5c	\	124	174	7c	
29	35	1d	GS (group separator)	61	75	3d	=	93	135	5d	]	125	175	7d	}
30	36	1e	RS (record separator)	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	US (unit separator)	63	77	3f	?	95	137	5f	_	127	177	7f	DEL (delete)

Dati testuali

Unicode - UTF8

यूनिकोड (Unicode), प्रत्येक अक्षर के लिए एक विशेष संख्या प्रदान करता है, चाहे कोई भी कम्प्यूटर प्लेटफॉर्म, प्रोग्राम अथवा कोई भी भाषा हो। यूनिकोड स्टैंडर्ड को एपल, एच.पी., आई.बी.एम., जस्ट सिस्टम, माइक्रोसॉफ्ट, ऑरेकल, सैप, सन, साईबेस, यूनिसिस जैसी उद्योग की प्रमुख कम्पनियों और कई अन्य ने अपनाया है। यूनिकोड की आवश्यकता आधुनिक मानदंडों, जैसे एक्स.एम.एल, जावा, एकमा स्क्रिप्ट (जावास्क्रिप्ट), एल.डी.ए.पी., कोर्बा 3.0

Unicode, 中文又稱萬國碼、國際碼、統一碼、單一碼, 是電腦科學領域裡的一項業界標準。它對世界上大部分的文字系統進行了整理、編碼, 使得電腦可以用更為簡單的方式來呈現和處理文字。

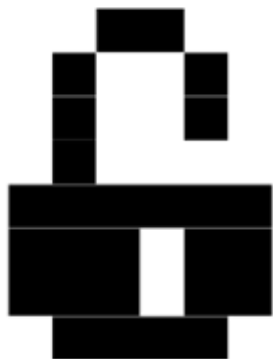
ကုန်ပျူတာတွင် ယူနီကုဒ်သည် ကမ္ဘာပေါ်ရှိ စာရေးစနစ် (writing system) အားလုံးနီးပါး ပါရှိပြီး ကုန်ပျူတာများနှင့် လိုက်ရောညီထွေ ရှိစေရေးအတွက် သက်မှတ်ထားသော စက်မှုစံ (industry standard) တစ်ခုဖြစ်သည်။ Universal Character Set စံနှင့်အတူ ယူနီကုဒ်သည် အက္ခရာပေါင်း ၁၀၀ ၀၀၀ ကျော်ပါသော စာအုပ် ထုတ်ဝေခဲ့သည်။ ထိုစံစာအုပ်တွင် ကိုးကားရန် ဇယားများ၊ encoding နည်းစဉ်များ၊ character encoding စံများ၊ အက္ခရာ၏ သဘောသဘာဝများ (ဥပမာ စာလုံးကြီး၊ စာလုံးသေး)၊ အထောက်အကူပြု computer file များ၊ အခြား သက်ဆိုင်ရာများ (အက္ခရာသဘော၊ normalization ဥပဒေ၊ ခွဲခြင်း၊ ပေါင်းခြင်း၊ rendering နှင့် နှစ်ဖက်သွား စာများ၏ အစဉ်၊ ဘယ်ညာ သဘောများ စသည်) ပါရှိသည်။^[၁]

UTF-8																
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F
6	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F
8	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F
9	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	+10	+11	+12	+13	+14	+15	+16	+17	+18	+19	+1A	+1B	+1C	+1D	+1E	+1F
A	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	+20	+21	+22	+23	+24	+25	+26	+27	+28	+29	+2A	+2B	+2C	+2D	+2E	+2F
B	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	+30	+31	+32	+33	+34	+35	+36	+37	+38	+39	+3A	+3B	+3C	+3D	+3E	+3F
2	1	2	LATIN	LATIN	LATIN	LATIN	LATIN	LATIN	LATIN	IPA	IPA	IPA	ACCENTS	ACCENTS	GREEK	GREEK
C	0001	0002	0080	00C0	0100	0140	0180	01C0	0200	0240	0280	02C0	0300	0340	0380	03C0
2	CYRIL	CYRIL	CYRIL	CYRIL	CYRIL	ARMENI	HEBREW	HEBREW	ARABIC	ARABIC	ARABIC	ARABIC	SYRIAC	ARABIC	THAANA	N'Ko
D	0400	0440	0480	04C0	0500	0540	0580	05C0	0600	0640	0680	06C0	0700	0740	0780	07C0
3	INDIC	MISC.	SYMBOL	KANA...	CJK	CJK	CJK	CJK	CJK	CJK	ASIAN	HANGUL	HANGUL	HANGUL	PUA	FORMS
E	0800	1000	2000	3000	4000	5000	6000	7000	8000	9000	A000	B000	C000	D000	E000	F000
4	SMP...	□	□	SSP...	SPU...	1	2	3	4	5	6	7	8	9		
F	10000	40000	80000	C0000	100000	110000	120000	130000	140000	150000	160000	170000	180000	190000		

Immagini raster (bitmap)

Un'immagine in bianco e nero può essere codificata decidendo che, per esempio, i pixel neri valgono 1, quelli bianchi valgono 0.

Con questa regola, siamo tornati ai numeri binari, che sappiamo trattare!



0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	0	1	1	0
0	1	1	1	0	1	1	0
0	0	1	1	1	1	0	0



0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	0	1	1	0
0	1	1	1	0	1	1	0
0	0	1	1	1	1	0	0

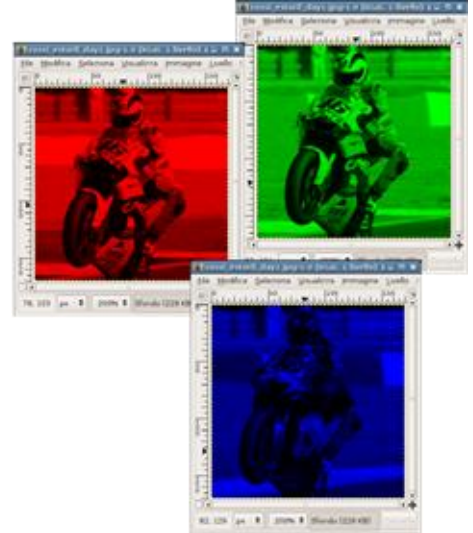


hex	dec
18	24
24	36
24	36
20	32
7E	126
76	118
76	118
3C	60

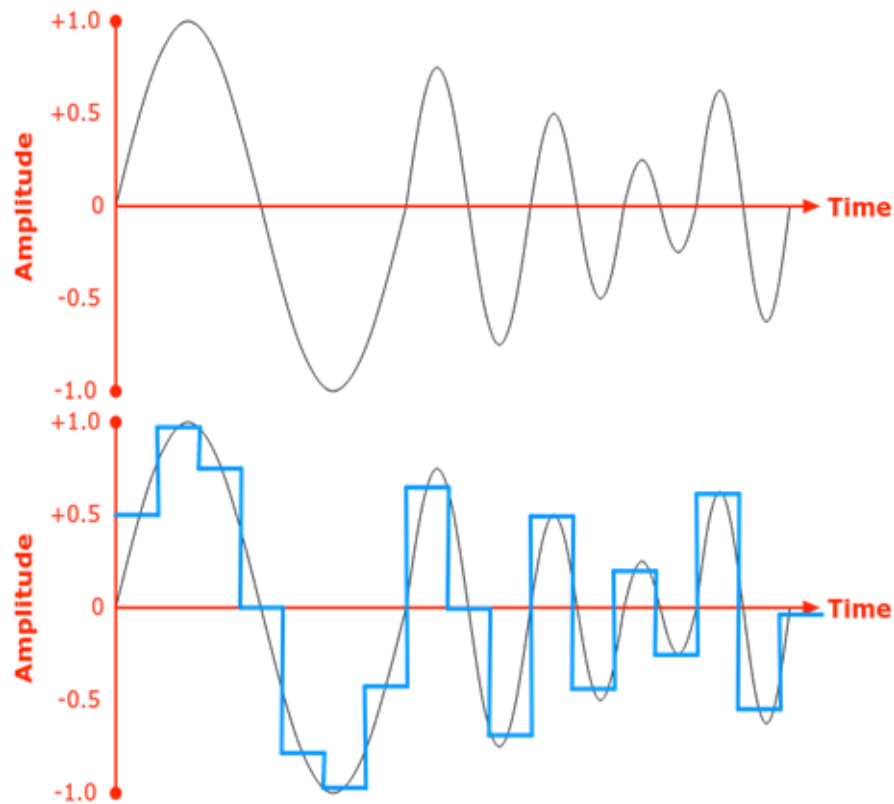
Immagini raster (a scala di grigio e a colori)

Possiamo codificare le immagini a scala di grigio, assegnando un valore numerico ad ogni pixel, per esempio: 0=nero, 255=bianco, con valori intermedi che rappresentano i grigi.

Le immagini a colori possono essere codificate con tre immagini a scala di grigio (una rappresenta l'intensità del rosso, una del verde, una del blu)



Suono campionato



Dal punto di vista fisico, il suono è una variazione di pressione nel tempo

Possiamo approssimare la curva “vera” dell’ampiezza misurandola in certi momenti specifici; questa operazione è detta *campionamento*.

Le misure presa con il campionamento sono una sequenza di numeri → codifica!

In pratica, si misura da -128 a +127 oppure -32768 a +32767 anziché da -1.0 a +1.0, così si lavora con interi in complemento a 2

Operatori binari logici

Operatore	Nome	Descrizione	Esempio
$a \ \& \ b$	AND	Data una specifica posizione pone il bit a 1 se i bit di entrambe le variabili sono 1	$0101 \ \& \ 1100 = 0100$
$a \ \ b$	OR	Data una specifica posizione pone il bit a 1 se almeno un bit è pari a 1	$0101 \ \ 1100 = 1101$
$a \ ^ \ b$	XOR	Data una specifica posizione pone il bit a 1 se uno e un solo bit è pari a 1	$0101 \ ^ \ 1100 = 1001$
$\sim a$	NOT	Inverte tutti i bit	$\sim 0101 = 1010$

Operatori binari di scorrimento (Bit Shift)

1 spostato
0 perso
0 nuovo
1 copiato

Operatore	Nome	Descrizione	Esempio
$n \ll x$	Bit shift a sx	Sposta i bit di n a sinistra di x posizioni, inserendo 0 a destra	0101 $\ll$ 1 = 1010
$n \gg x$	Bit shift a dx con mantenimento del segno	Sposta i bit di n a destra di x posizioni, copiando il <u>bit di segno</u> a sinistra	0101 $\gg$ 1 = 0010 1101 $\gg$ 1 = 1110
$n \ggg x$	Bit shift a dx con inserimento di 0	Sposta i bit di n a destra di x posizioni, inserendo 0 a sinistra	0101 $\ggg$ 1 = 0010 1101 $\ggg$ 1 = 0110

Q & A