

## Binary System

Il sistema numerico binario

- base 2

- usa solo due simboli, di solito 0 1

Quello decimale  $\rightarrow$  base 10  $\rightarrow$  10 cifre

È il sistema usato in informatica (nei computer) per la rappresentazione interna dell'informazione perché come visto nei circuiti digitali (tipo Flip-Flop) è molto conveniente la gestione dell'informazione con due soli valori (livelli di tensione/corrente elettrica). Vero-falso logico booleano.

Vediamo adesso con esempi come gestisco in binario

- un numero decimale
- il suo segno
- le operazioni somme, sottrazioni
- ottali / esadecimali

Vediamo esempi

• Conversione da base 2 a base 10:

Prendiamo in base 10  $237 = 2 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0$

Analogamente in base 2  $11010010 =$   
 $= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 210$

128
64
16
2

• Viceversa: Conversione base 10 → base 2

145					→ divido per due e
7	2	1	(LSB)		scrivo sotto valore e resto
3	6	0		↙	Bit meno significativo - Least Significant Bit
1	8	0			Il primo resto che ottengo mi dice
	9	0			semplicemente se il numero è
	4	1			pari 0 o dispari 1
	2	0			
	1	0			
0	1	(MSB)	bit più significativo		

otengo quindi la sequenza inversa di bit in binario  
quindi il numero binario si scrive dal MSB al LSB

MSB								LSB
1	0	0	1	0	0	0	0	1
$2^7$	+	$2^4$		+	$2^0$	=	145	

- Se uso 8 bit

da	0	0	0	0	0	0	0	0
a	1	1	1	1	1	1	1	1

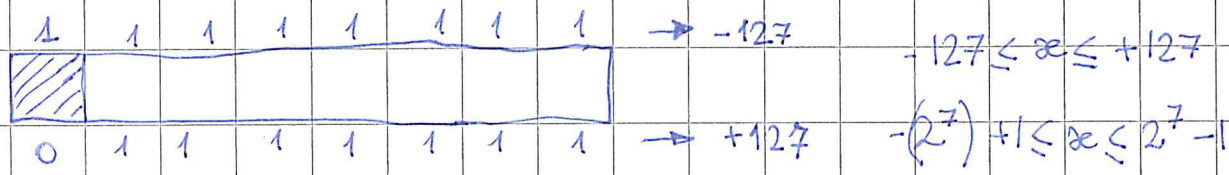
255

Ad esempio ogni casella di memoria è un flip-flop che ha come possibili valori 0 e 1  
Posso rappresentare 256 numeri naturali (solo positivi) <sup>per ora</sup>

- Come faccio per i numeri negativi?

Prendo il MSB e lo riservo per descrivere il segno 0 = + e 1 = -

In questo caso su 8 bit: 7 per il numero e 1 per il segno  
da 0 a 127



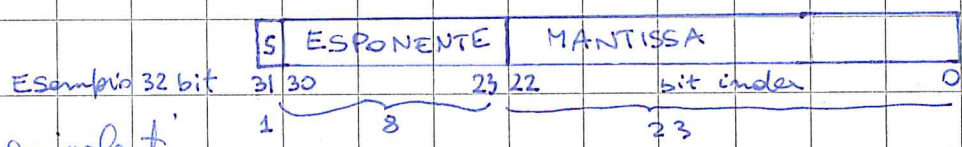
In totale ho 255 numeri x -127 < x < +127  
ho due modi di scrivere lo zero +0 e -0

Se invece di avere 8 bit, ne avessi 16	± 2 <sup>15</sup>	(new 1)
A 32 bit	± 2 <sup>31</sup>	2 · 10 <sup>9</sup>
A 64 bit	± 2 <sup>63</sup>	9 · 10 <sup>18</sup>
		± 65 535

I computer <sup>rappresentano</sup> trattano i numeri razionali suddividendoli in

MANTISSA × BASE      ESPONENTE      a = M × b<sup>E</sup>

ES. 1.2345 = 1.2345 × 10<sup>-4</sup>



- Nomi alternativi equivalenti
- Mantissa o significando; in inglese { significant / mantissa / coefficient
- Espoente o caratteristica; " " exponent

Oss. Passando da un intero a un razionale perdo informazioni  
Posso "rigneddegnarla" usando double precision  
64 bit invece di 32 bit      S = 1      E = 11      M = 52

# Rappresentazione in OTTALI

base 8 → uso le cifre da 0 a 7

raccoglie i bit in gruppi di 3

binario 1 0 0 1 0 0 0 1 = decimale 145

(0 1 0) (0 1 0) (0 0 1)

ottale 2 2 1 = ottale 221

I primi computer avevano sistemi di reiniziazione delle parole di 6, 12, 24 bit tutti divisibili per 3 e usavano la rappresentazione ottale

Ad esempio CAMAC 24 bit

# Rappresentazione esadecimale EXADECIMAL HEX

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

→ raccoglie i bit in gruppi di 4

145 decimale = binario 1 0 0 1 0 0 0 1

Hex

9

1

# Binary and Hexadecimal System

Decimal System

$$237 = 2 * 10^2 + 3 * 10^1 + 7 * 10^0$$

Conversion: Base 2 → Base 10

$$11010010 =$$

$$1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 210$$

Conversion: Base 10 → Base 2

145 | → divide 145:2=72 with remainder of 1...

72		1 (least significant digit)
36		0
18		0
9		0
4		1
2		0
1		0
0		1 (most significant digit)

145 in base 10 corresponds to 10010001 in base 2 with N bits we can represent all integer numbers between 0 and 2^N-1

DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Hex Bin

0	0000	Conversion: Base 16 → Base 2:
1	0001	Hexadecimal number 3F5 corresponds to 0011 1111 0101
2	0010	
3	0011	

Conversion Base 2 → Base 16:

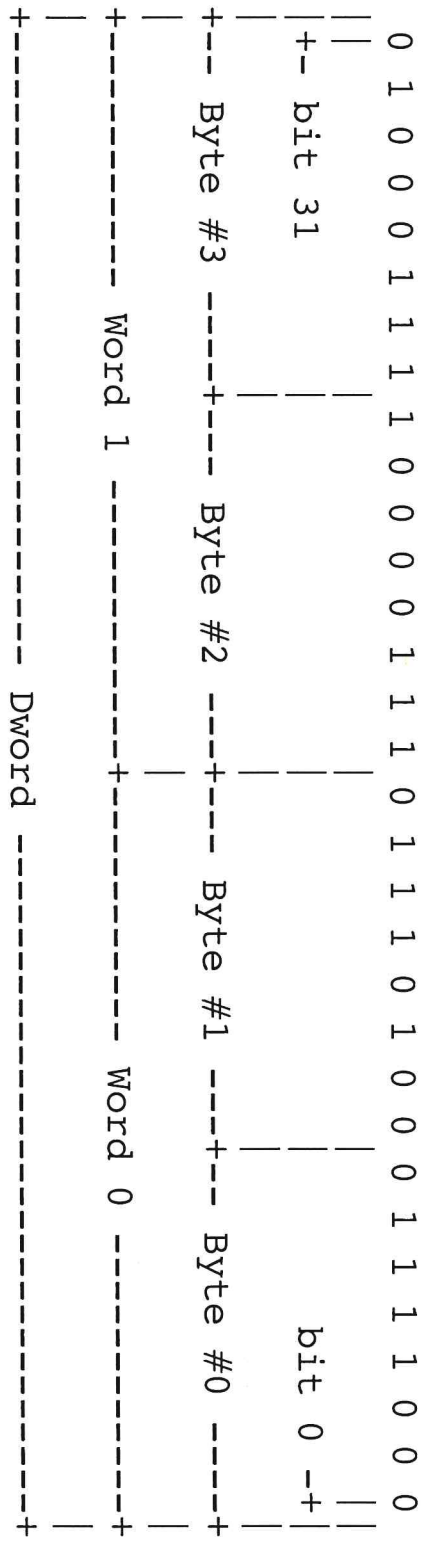
4	0100	10 0101 1101 0101 0010
5	0101	
6	0110	
7	0111	
8	1000	0010 → 2
9	1001	0101 → 5
A	1010	1101 → D
B	1011	10 → 2
C	1100	
D	1101	0010 0101 1101 0101 0010 corresponds to
E	1110	2 5 D 5 2
F	1111	

*Operatori logici bit per bit*

# Bitwise Operators

Bitwise operators work on binary data

1 Byte = 8 bits; 1 Word = 2 Bytes = 16 bits; *Double word* Dword = 2 Words = 32 bits



These are the 6 most important Bitwise operators:

- 1 & AND operator
- 2 | OR operator
- 3 ^ XOR operator
- 4 ~ Ones Complement (Bitwise Not)
- 5 >> Right Shift operator
- 6 << Left Shift operator

# Bitwise Operators

## 1 AND

1	&	1	==	1
1	&	0	==	0
0	&	1	==	0
0	&	0	==	0

00110010

& 00010000

-----  
00010000

## 3 XOR

1	^	1	==	0
1	^	0	==	1
0	^	1	==	1
0	^	0	==	0

00110010

^ 00011000

-----  
00101010

## 5 LEFT SHIFT

00001100 << 2 (decimal) 11111111  
 -----  
 00110000 11111100

*SHIFT : shifto tutti i bit  
rigidamente  
altro esempio*

## 6 RIGHT SHIFT

00001100 >> 2 (decimal) 11111111  
 -----  
 00000011 00111111

## 2 OR

1		1	==	1
1		0	==	1
0		1	==	1
0		0	==	0

00110010

| 00000100

-----  
00110110

## 4 Bitwise NOT

~ 1	==	0
~ 0	==	1
~ 00000011		

-----  
111111100

## RIGHT AND LEFT SHIFT

(be careful...)

00001111 >> 2 (decimal)

-----  
00000011 << 2 (decimal)

-----  
00001100

00111100

Inoltre

**MASK**

se sono interessato solo ad alcuni bit

facio l'AND tra la <sup>word</sup> parte binaria e la <sup>mask</sup> maschera nella posizione dei 0 del bit che mi interessa:

A        1 0 1 1 0 1 0 1

MASK    0 0 0 0 0 1 0 0        mi interessa il 3° bit

A & MASK = 0 0 0 0 0 1 0 0

Con queste operazioni "bit wise" riesco a gestire facilmente le operazioni fra numeri positivi.

Per quelli negativi serve usare qualche

"trucco": Complemento (negazione) a 1 (aggiunta del bit perso).

ES1        3        0 0 0 0 0 0 1 1

          -1        1 0 0 0 0 0 0 1

anzichè -1 prendo il complemento a 1 di 1

finiamo di vedere le operazioni prox volta