CONVERSIONI BINARIO → DECIMALE

2^3 2^2 2^1 2^0

$1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8+2+1 = 11$

CONVERSIONE DECIMALE → BINARIO

27 : 2 = 13 (1)

13 : 2 = 6 (1)

6 : 2 = 3 (0)

3 : 2 = 1 (1)

1 : 2 = 0 (1)

16 8 4 2 1

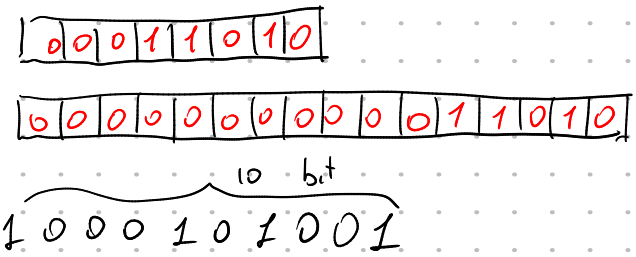
11011_2

$16+8+2+1 = 27$

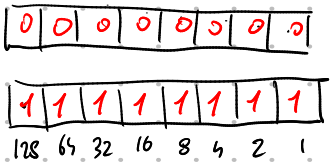
$11010_2 = 26_{10}$ 0026

byte → 1 byte = 8 bit

short → 2 byte = 16 bit



1000101001



minimo

massimo

$255 = 256-1 = 2^8 - 1$ (8)

somma 1

togliamo 1

100000000 = 256

256 128 1 1 1 (2 1)

short

minimo : 0

massimo : $2^{16}-1 = 65535$

unsigned byte b = 200;

b += 100;

128 64 32 16 8 4 2 1

200_{10} binario 11001000 +

100_{10} binario 01100100 =

100101100

32 16 8 4 2 1

9 bit

44 base 10

RAPPRESENTAZIONE DI INTERI POSITIVI E NEGATIVI

$-1011_2 = -11_{10}$ (+11)

byte: 1 0 0 0 1 0 1 1 (0)

bit più significativo
0 = positivo
1 = negativo

short: 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 (0)

RAPPRESENTAZIONE IN MODULO E SEGNO

minimo: 1 1 1 1 1 1 1 1 → −127 (127)

massimo: 0 1 1 1 1 1 1 1 → +127

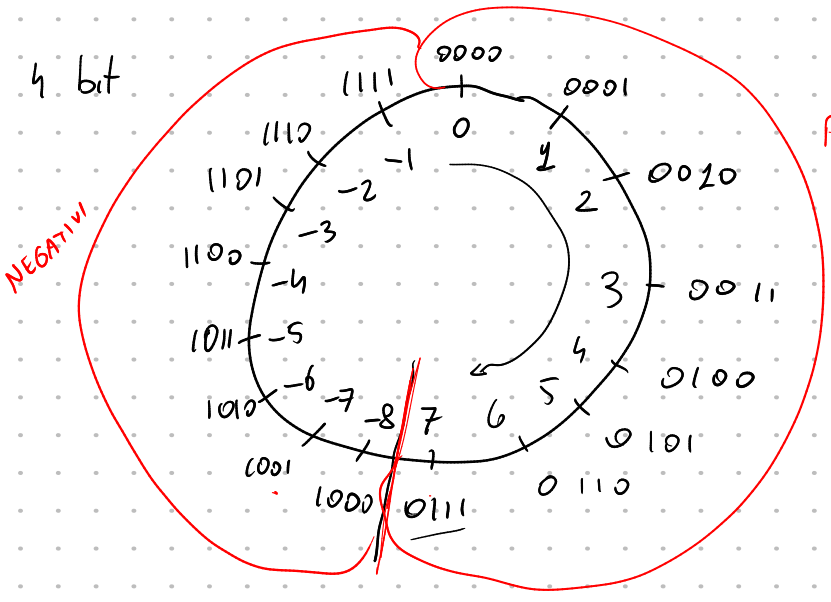
DIFETTI:

1° ci sono due zeri
00000000 +0
10000000 −0

2° difficile fare le operazioni

COMPLEMENTO A 2

4 bit



POSITIVI COME PER MODULO E SEGNO

su 8 bit, quanto vale $\boxed{0}\boxed{0001001}$? = 9

0 → positivo; 0001001 → valore assoluto

$\boxed{1}\boxed{0001001}$?

1 → negativo

complemento a 1: scambio tutti gli 1 con 0 e viceversa

1110110

↓ +1

$\boxed{1110111}$ (64 32 16 8 4 2 1) → base 10 → 119_{10}

-119

Conversione da decimale a binario complemento a 2 (8 bit)

33_{10} → binario → (32 16 8 4 2 1) 100001_2

RAPPRES. IN COMPLEM. A 2 SU 8 bit DI 33 COINCIDE CON LA RAPPRES. IN MODULO E SEGNO

$\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}$

segno; 0 per occupare tutto lo spazio

-33 → binario → 100001_2 → allargo per occupare 7 bit → 0100001

→ complemento a 1 → 1011110 → +1 → 1011111

$\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}$

RAPPRESENTAZIONE SU 8 bit IN complemento a 2 del numero -33.

su 8 bit complemento a 2

- massimo: $01111111 = 127 = 2^7 - 1$
- minimo: $10000000 = -128 = -2^7$

per un tipo a 16 bit (short)

- massimo $= 2^{15} - 1 = 32767$
- minimo $= -2^{15} = -32768$

Perché la rappresentazione in complemento a 2 è meglio di quella modulo e segno:

1) C'è un solo zero
2) c'è un unico algoritmo per la somma che si applica sempre

$123 + 14$ — ALGORITMO DI SOMMA

$123 + (-14)$ — ALGORITMO PER LA DIFFERENZA

} Gli esseri umani usano due algoritmi per le somme degli interi

$100 + (-33)$

-33 → complemento a 2 → 11011111

100 → complemento a 2 → 01100100

$11011111 + 01100100 = 101000011$ (vedi sopra per i calcoli)

segno = 1 → negativo

01000011: $64 + 2 + 1 = 67$

67

CASO DI ERRORE (OVERFLOW)

-33 → compl. a 2 → 11011111 (riporti: 1 1 1) +

-100 → complement a 2 → 10011100

01111011

$64\ 32\ 16\ 8\ 2\ 1 = 123$

-100

↓

1100100

↓ espanderlo se < bit
ma è già se > bit

1100100

↓ complement a 1

$00110111 +$
$\quad\quad\quad 1$

0011100

RAPPRESENTAZIONE DEI CARATTERI

SERVE UNA TABELLA DI CONVERSIONE DA NUMERI A CARATTERI

IN ORIGINE C'ERANO DUE TABELLE DI CONVERSIONE STANDARD

ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

~~EBCDIC~~

limitata ai caratteri dell'alfabeto latino, più le cifre 0–9, più qualche altra cosa

Usa solo 7 bit: i codici ASCII vanno da 0 a 127

ASCII

- LATIN-1: ASCII + CARATTERI ACCENTATI
- LATIN-7: ASCII + ALFABETO GRECO
- GIAPPONESE
- CINESE

UNICODE 32 bit — $0 \dots 2^{32} - 1$

4 miliardi e qualcosa

DIVERSE RAPPRESENTAZIONI DELLA STESSA TABELLA

- UTF-32 effettivamente uso 32 bit per ogni carattere
- UTF-8 codice a lunghezza variabile dove i caratteri possono occupare 8, 16, 24 o 32 bit (LINUX, MAC)
- UTF-16 codice a lunghezza variabile ma solo due casi: 16 o 32 bit.

JAVA USA UTF-16 ma solo nel caso in cui il carattere occupa 16 bit.