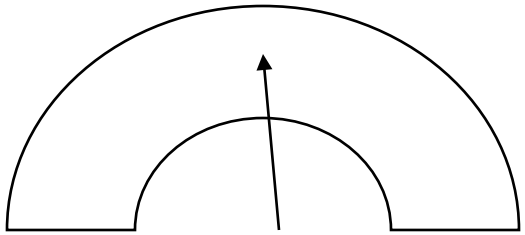
Programma del corso

- *Introduzione agli algoritmi*
- ***Rappresentazione delle Informazioni***
- *Architettura del calcolatore*
- *Reti di Calcolatori (Reti Locali, Internet)*
- *Elementi di Programmazione*

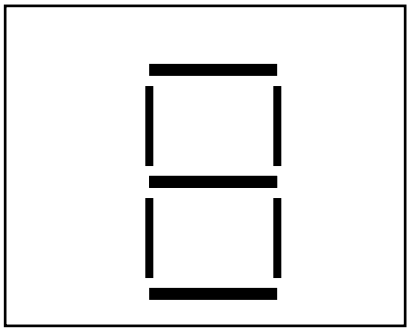
Rappresentazione dell'informazione

- Varie rappresentazioni sono possibili per la medesima informazione
  - Es. Testo scritto su carta o registrato su nastro
- Due rappresentazioni R1 e R2 sono equivalenti se data R1 è possibile ricavare R2 e viceversa
  - Es. Trascrizione del testo data la sua registrazione e viceversa
- Scelta della rappresentazione
  - Spesso convenzionale ...
  - ... ma spesso legata a vincoli
  - Es. Rappresentazione binaria negli elaboratori

Analogico vs digitale



Informazione esplicita nel supporto: per analogia

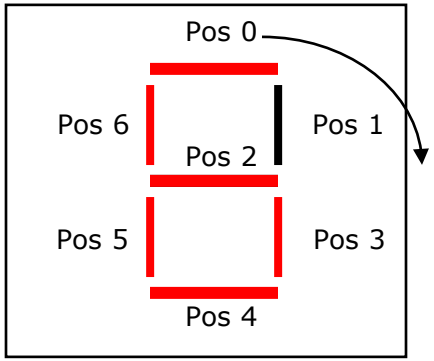


Informazione implicita nella rappresentazione: serve codifica/decodifica

Il numero 6 si codificherebbe come

1011111

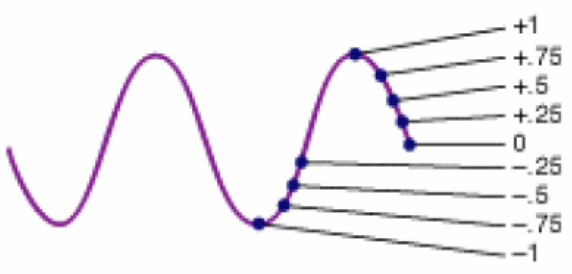
acceso spento



La rappresentazione digitale dell'informazione

- Rappresentazione digitale
  - Ogni dato viene codificato impiegando entità distinte individualmente e organizzate in modo opportuno (es. abaco). Trova le sue origini nel conteggio con le *dita della mano* (da cui il nome).

- Rappresentazione analogica
  - Basata sull'impiego di dispositivi che realizzano una grandezza fisica che *può variare in modo continuo* (es. tensione elettrica).



- Calcolatori analogici e digitali
  - Durante questo secolo sono stati sviluppati sia calcolatori analogici che digitali ma, la rappresentazione che si è affermata è di tipo *digitale*.

Dalla rappresentazione alla codifica dell'informazione

rappresentazione

- Alfabeto: insieme di simboli
  - Es. le 10 cifre (da 0 a 9)
- Stringhe: concatenazioni si simboli dell'alfabeto
  - Es. la stringa 123
  - Esiste un insieme di configurazioni possibili (di solito finito)
- Processo di codifica: da informazione a una stringa che la rappresenta
  - Convenzionale: deve essere condiviso da chi usa
- Processo di decodifica: da una stringa ad informazione

Codifica dell'informazione

- Il calcolatore memorizza ed elabora vari tipi di informazioni
  - Numeri, testi, immagini, suoni
- Occorre rappresentare tale informazione in formato facilmente manipolabile dall'elaboratore

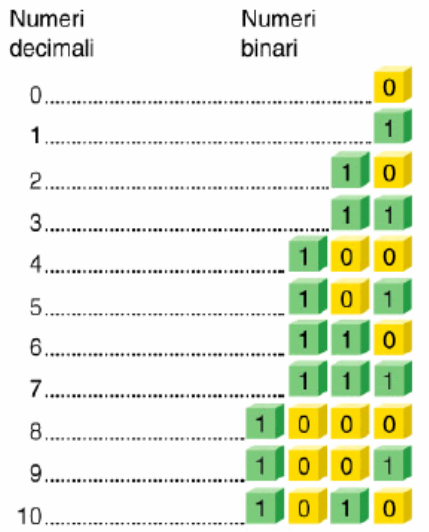
L’aritmetica dei Calcolatori

- L’aritmetica usata dai calcolatori è diversa da quella comunemente utilizzata dalle persone.
- *La precisione* con cui i numeri possono essere espressi è *finita* e predeterminata poiché questi devono essere memorizzati entro un limitato spazio di memoria.

$$\sqrt{2} = \boxed{1}\boxed{.}\boxed{1}\boxed{4}\boxed{1}\boxed{4}\boxed{2}\ 1\ 3\ 5\ 6$$

- *La rappresentazione* è normalmente ottenuta utilizzando il *sistema binario* poiché più adatto a essere maneggiato dal calcolatore.

124 ➔ 01111100



Rappresentazione delle informazioni

Idea di fondo

- usare presenza/assenza di carica elettrica
- usare passaggio/non passaggio di corrente/luce

Usiamo cioè una rappresentazione binaria (a due valori) dell'informazione

L'unità minimale di rappresentazione è il **BIT** (**BI**nary digi**T** – cifra digitale): **0** o **1**

Informazioni complesse

Con 1 bit rappresentiamo solo 2 diverse informazioni:

si/no - on/off - 0/1

Mettendo insieme più bit possiamo rappresentare più informazioni:

00 / 01 / 10 / 11

Informazioni complesse si memorizzano come sequenze di bit

Informazioni complesse

- Per codificare i nomi delle 4 stagioni bastano 2 bit

- Ad esempio:
  - **0 0** per rappresentare **Inverno**
  - **0 1** per rappresentare **Primavera**
  - **1 0** per rappresentare **Estate**
  - **1 1** per rappresentare **Autunno**

- Quanti bit per codificare i nomi dei giorni della settimana?

Informazioni complesse

In generale, con **N** bit, ognuno dei quali può assumere **2** valori, possiamo rappresentare 2^N informazioni diverse (**tutte le possibili combinazioni di 0 e 1 su N posizioni**)

viceversa

Per rappresentare **M** informazioni dobbiamo usare **N** bit, in modo che $2^N >= M$

Esempio

Per rappresentare **57** informazioni diverse dobbiamo usare gruppi di almeno **6** bit. Infatti:

$$2^6 = 64 > 57$$

Cioè un gruppo di 6 bit può assumere 64 configurazioni diverse:
000000 / 000001 / 000010 …/ 111110 / 111111

Il Byte

- Una sequenza di **8 bit** viene chiamata ***Byte***
  - 0 0 0 0 0 0 0 0
  - 0 0 0 0 0 0 0 1
  - .....

byte = 8 bit = 2^8 = 256 informazioni diverse

Usato come unità di misura per indicare

- le dimensioni della memoria
- la velocità di trasmissione
- la "potenza" di un elaboratore

Usando sequenze di byte (e quindi di bit) si possono rappresentare caratteri, numeri immagini, suoni.

Altre unità di misura

- KiloByte (**KB**), MegaByte (**MB**), GigaByte (**GB**), …
- Per ragioni storiche in informatica Kilo, Mega, e Giga indicano però le ***potenze di 2*** che più si avvicinano alle corrispondenti potenze di 10 (Sistema IEC)
- Sistema SI: 1 Kilobyte = 1000 byte
- Sistema IEC: 1 Kilobyte (detto *Kibibyte* = 1024 byte)

- Più precisamente (sistema IEC)
  - 1 KB = 1024 x 1 byte = $2^{10} \sim 10^3$ byte
  - 1 MB = 1024 x 1 KB = $2^{20} \sim 10^6$ byte
  - 1 GB = 1024 x 1 MB = $2^{30} \sim 10^9$ byte

- Il sistema IEC è usato come unità di misura per la capacità della memoria di un elaboratore.
- Il sistema SI è usato come unità di misura per le capacità degli hard disk

Unità di misura nel sistema binario

- Il bit rappresenta la più piccola unità di misura dell'informazione memorizzabile in un calcolatore. I sistemi moderni memorizzano e manipolano miliardi di bit; per questo motivo sono stati definiti diversi multipli.

Nome	Sigla	In bit	In byte	In potenze di 2
Bit	Bit	1 bit	1/8	2^1=2 stati
Byte	Byte	8	1	2^8=256 stati
KiloByte	KB	8.192	1.024	2^{10} Byte
MegaByte	MB	8.388.608	1.048.576	2^{20} Byte
GigaByte	GB	8.589.934.592	1.073.741.824	2^{30} Byte
TeraByte	TB	8.796.093.022.208	1.099.511.627.776	2^{40} Byte

ATTENZIONE: 1KB non corrisponde a 1000 Byte, ma a 1024 Byte, 1MB non corrisponde a 1000000 Byte, ...

Codici per i simboli dell'alfabeto

- Per rappresentare i simboli dell'alfabeto anglosassone (0 1 2 ... A B ... a b ...) bastano 7 bit (codifica **ASCII**)
  - Nota: *B* e *b* sono simboli diversi
  - 26 maiuscole + 26 minuscole + 10 cifre + 30 segni di interpunzione+… -> circa 120 oggetti

- Per l'alfabeto esteso con simboli quali &, %, $, ... bastano 8 bit come nella codifica accettata universalmente chiamata **ASCII esteso**
- Per manipolare un numero maggiore di simboli si utilizza la codifica **UNICODE** a 16 bit

Codifica ASCII

- La codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nterchange **C**ode) utilizza codici su 7 bit
  (**2^7 = 128 caratteri diversi**)
- Ad esempio
  - 1 0 0 0 0 0 1 rappresenta A
  - 1 0 0 0 0 1 0 rappresenta B
  - 1 0 0 0 0 1 1 rappresenta C

- Le parole si codificano utilizzando sequenze di valori da 7 bit
  - 1000010 1000001 1000010 1000001
    B A B A

Altri codici di codifica

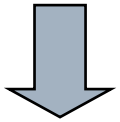
- **ASCII ESTESO**
  - Usa anche il primo bit di ogni byte
  - 256 caratteri diversi
  - non è standard (cambia con la lingua usata)
  - Ad es. a volte nello scambio di mail, ci si trova con strani caratteri (sono magari le lettere accentate non riconosciute dal programma di gestione delle mail)
- **ISO 8859-1: contiene i caratteri latini di maggior uso (coincide con ASCII per i primi 127 valori)**
- **UNICODE (UTF-8 e UTF-16)**
  - standard proposto a 8 e 16 bit (65.536 caratteri)
  - UTF-8 è usato per le e-mail
- **EBCDIC**
  - altro codice a 8 bit della IBM (quasi in disuso)

Tabella ASCII (0-127)

00000000	Null	00100000	Spc	01000000	@	01100000	`
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	01000101	E	01100101	e
00000110	Acknowledge	00100110	&	01000110	F	01100110	f
00000111	Audible bell	00100111	'	01000111	G	01100111	g
00001000	Backspace	00101000	(	01001000	H	01101000	h
00001001	Horizontal tab	00101001	)	01001001	I	01101001	i
00001010	Line feed	00101010	*	01001010	J	01101010	j
00001011	Vertical tab	00101011	+	01001011	K	01101011	k
00001100	Form Feed	00101100	,	01001100	L	01101100	l
00001101	Carriage return	00101101	-	01001101	M	01101101	m
00001110	Shift out	00101110	.	01001110	N	01101110	n
00001111	Shift in	00101111	/	01001111	O	01101111	o
00010000	Data link escape	00110000	0	01010000	P	01110000	p
00010001	Device control 1	00110001	1	01010001	Q	01110001	q
00010010	Device control 2	00110010	2	01010010	R	01110010	r
00010011	Device control 3	00110011	3	01010011	S	01110011	s
00010100	Device control 4	00110100	4	01010100	T	01110100	t
00010101	Neg. acknowledge	00110101	5	01010101	U	01110101	u
00010110	Synchronous idle	00110110	6	01010110	V	01110110	v
00010111	End trans. block	00110111	7	01010111	W	01110111	w
00011000	Cancel	00111000	8	01011000	X	01111000	x
00011001	End of medium	00111001	9	01011001	Y	01111001	y
00011010	Substitution	00111010	:	01011010	Z	01111010	z
00011011	Escape	00111011	;	01011011	[	01111011	{
00011100	File separator	00111100	<	01011100	\	01111100	|
00011101	Group separator	00111101	=	01011101	]	01111101	}
00011110	Record Separator	00111110	>	01011110	^	01111110	~
00011111	Unit separator	00111111	?	01011111	_	01111111	Del

Numeri in ASCII

Le cifre 0..9 rappresentate in Ascii sono simboli o caratteri **NON** quantità numeriche



Non possiamo usarle per indicare quantità e per le operazioni aritmetiche. (Anche nella vita di tutti giorni usiamo i numeri come simboli e non come quantità: i n. telefonici)

Il sistema decimale

- 10 cifre di base: 0, 1, 2, …, 9
- **Notazione posizionale:** la posizione di una cifra in un numero indica il suo **peso** in potenze di **10.** I pesi sono:
  - Unità = $10^0 = 1$ (posiz. 0-esima)
  - decine = $10^1 = 10$ (posiz. 1-esima)
  - centinaia = $10^2 = 100$ (posiz. 2-esima)
  - migliaia = $10^3 = 1000$ (posiz. 3-esima)
  - … .. ..

Esempio di numero rappresentato in notazione decimale

Il **numerale** 2304 in notazione decimale (o in base 10) rappresenta la quantità:

$2304 = 2*10^3 + 3*10^2 + 0*10^1 + 4*10^0 =$

$2000 + 300 + 0 + 4 = 2304$ (**numero**)

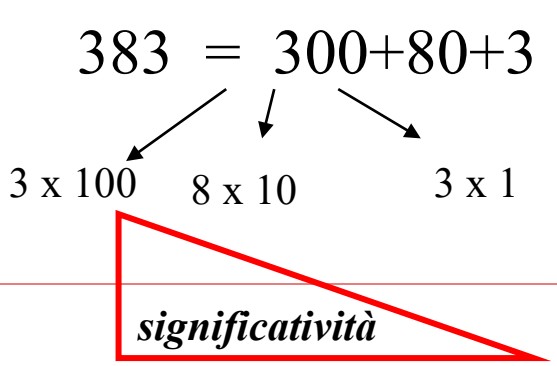
Nota: numero e numerale qui coincidono, perché il sistema decimale è quello adottato come sistema di riferimento

- NOTA: lo stesso numero è rappresentato da numerali diversi in diversi sistemi
  - **156** nel sistema decimale
  - **CLVI** in cifre romane

I sistemi di numerazione

POSIZIONALI

- ai diversi simboli dell'alfabeto (cifre), viene associato un valore crescente in modo lineare da destra verso sinistra;
- il significato di un simbolo (il suo valore) dipende ordinatamente dalla sua posizione nella stringa
- ESEMPIO: il sistema di numerazione decimale arabo: 10 simboli (0, 1, 2, ...9)



NON POSIZIONALI

- Il significato dei simboli non dipende dalla loro posizione
- ma è stabilito in base ad una legge additiva dei valori dei singoli simboli (se posti in ordine crescente)
- ESEMPIO: il sistema di numerazione romano

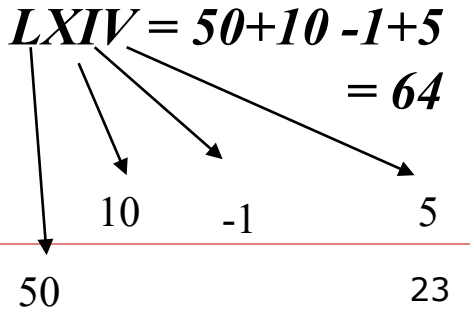
I = 1

V = 5

X = 10

L = 50

...



Notazione posizionale (generale)

Dato un numero a, tale valore è ottenuto:

$$a = \sum_{i=0}^{M-1} c_i b^i$$

Dove M è il numero di cifre di cui è composto, c_i il valore delle cifre e b la base del sistema posizionale.

Notazione posizionale (decimale)

Dato un numerale espresso come:

- $c_n c_{n-1} \ldots c_1 c_0$
  - dove i coefficienti c_i possono essere le cifre da 0 a 9

Il numero corrispondente è:

- $c_n * 10^n + c_{n-1} * 10^{n-1} + \ldots + c_1 * 10^1 + c_0 * 10^0$

- In base 10 con N cifre posso rappresentare i 10^N numeri da 0 a $10^N - 1$

Notazione posizionale (binaria)

- Considerando B=2
- Dato il numerale:
  - $c_n c_{n-1} \ldots c_1 c_0$
  - dove i coefficienti c_i possono essere 0 o 1
  - Il numero è: $c_n*2^n + \ldots + c_2*2^2 + c_1*2^1 + c_0*2^0$
- con N cifre riesco a rappresentare i 2^N numeri da 0 a 2^N-1

Esempio di numero rappresentato in notazione binaria

Conversione da binario a decimale:

dato ad esempio la sequenza 1101:

$$1101 = \sum_{i=0}^{3} c_i \, 2^i = 1 \text{ x } 2^3 + 1 \text{ x } 2^2 + 0 \text{ x } 2^1 + 1 \text{ x } 2^0 = 13$$

essendo che si potrebbe creare confusione tra 11 in binario e 11 in decimale, utilizzeremo il pedice per indicare il sistema di numerazione:

11_2 11_{10}

Il numero più grande rappresentato con N cifre

- Sist. Decimale = 99…99 = 10^N - 1
- Sist. Binario= 11..11 = 2^N - 1
- **Esempio**: 11111111 (8 bit binari) = 2^8 -1 = 255. Per rappresentare il n. 256 ci vuole un bit in più: 100000000 = $1*2^8$ = 256.

Quindi...

Fissate quante cifre (bit) sono usate per rappresentare i numeri, si fissa anche il numero più grande che si può rappresentare:

- con 16 bit: $2^{16} - 1 = 65.535$
- con 32 bit: $2^{32} - 1 = 4.294.967.295$
- con 64 bit: $2^{64} - 1 =$ circa $1,84 * 10^{19}$

Sistema di numerazione binario

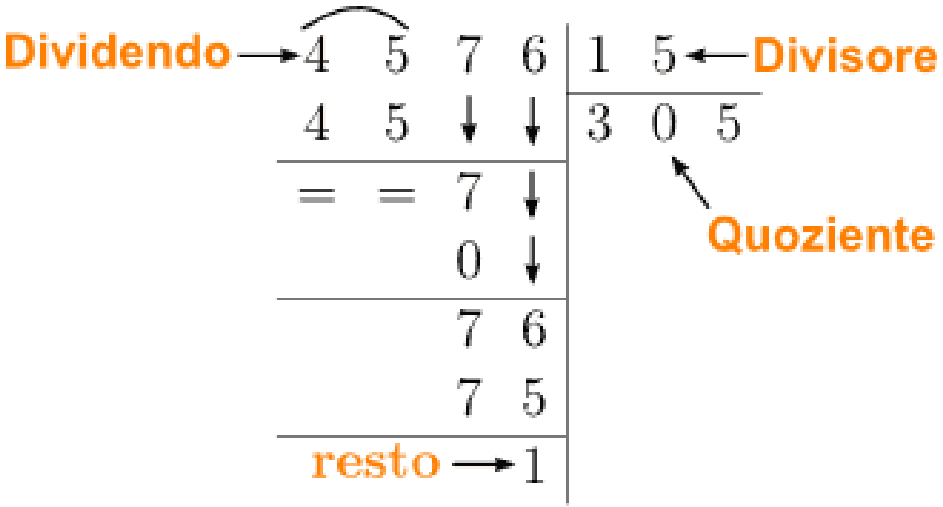
Conversione da decimale a binario

Per la conversione di un numero naturale dal sistema decimale al binario si utilizza un metodo iterativo che sfrutta l'algoritmo della **divisione euclidea**.

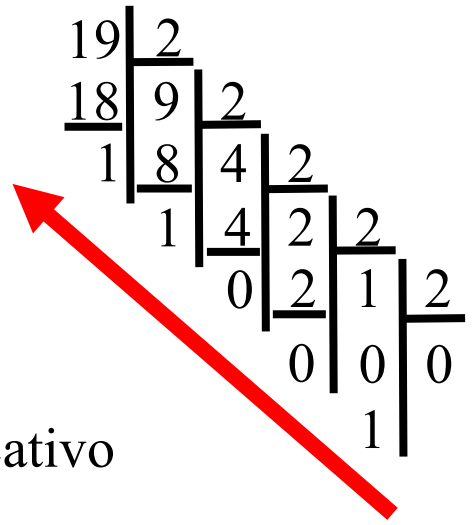
La prima divisione si effettua considerando come dividendo il numero decimale da convertite e come divisore la base del sistema binario (cioè 2).

Se il quoziente è diverso da 0 (zero) si effettua un'ulteriore divisione per 2 in cui il nuovo dividendo è il quoziente appena ottenuto. In caso contrario il processo iterativo si conclude e il risultato della conversione è il numero binario che si ottiene considerando i resti delle divisioni effettuate a cominciare dall'ultima.

Sistema di numerazione binario



Sistema di numerazione binario



Valore meno significativo

Valore più significativo

$$19_{10} = 10011_2$$

$$10011_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= 16 + 2 + 1 = 19$$

Sistema di numerazione binario

Operazione di somma

$$
\begin{array}{r}
1111 \\
11011 + \\
00101 = \\
\hline
100000
\end{array}
$$

$$1_2 + 1_2 = 10_2$$

$$11011_2 = 27_{10},\ \ 00101_2 = 5_{10},\ 100000_2 = 32_{10}$$

Esistono anche altre basi di numerazione

CODICE OTTALE

- cifre: 0, 1, 2, 3, 4, 5, 6, 7
- 10 (ottale) = 8 (decimale)

CODICE ESADECIMALE

- cifre: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- 10 (esadecimale) = 16 (decimale); B = 11; $2B=2*16^1+B*16^0 = 32+11 = 43$

Aritmetica binaria

- Somma tra numeri binari

+	0	1
0	0	1
1	1	10

Somma tra numeri binari: alcuni esempi

$$\begin{array}{r} 1\ + \\ 1\ = \\ \hline 1\ 0 \end{array}$$

$$\begin{array}{r} 1\ 0\ 1\ + \\ 1\ 1\ = \\ \hline 1\ 0\ 0\ 0 \end{array}$$

$$\begin{array}{r} 1\ 1\ 0\ 1\ 0\ + \\ 1\ 0\ 1\ = \\ \hline 1\ 1\ 1\ 1\ 1 \end{array}$$

Rappresentazione di numeri positivi e negativi

- Il bit più a sinistra rappresenta il segno del numero:

  0 = '+' 1 = '-'

  1101 = -5

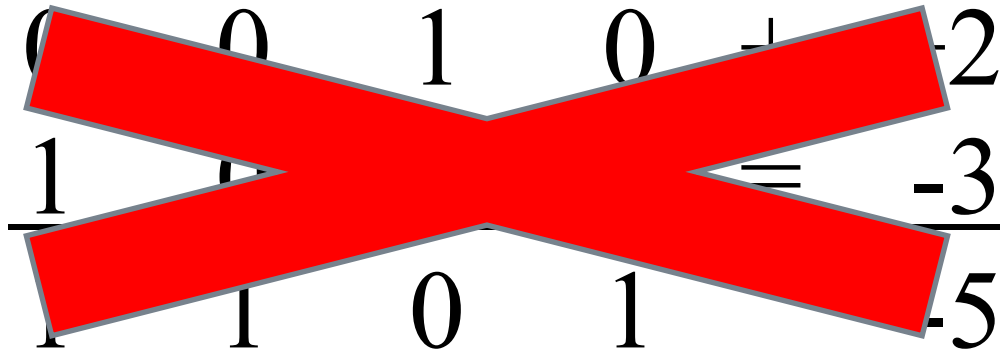
- E' indispensabile indicare il numero **N** di bit utilizzati:
  - **1** bit per il segno e **N-1** bit per il modulo
- Con un byte possiamo rappresentare tutti i numeri compresi tra

  +127 (01111111) e –127 (11111111)
  - In generale con **N** bit si rappresentano i valori da

    $-2^{N-1}-1$ a $+2^{N-1}-1$

Codifica dei numeri interi negativi

- Es: Due controindicazioni:
  - 2 rappresentazioni dello 0
  - non si possono applicare le regole tradizionali per le operazioni aritmetiche:



Rappresentazione di numeri positivi e negativi
Complemento a 2 (consultare Dispensa)

- Definizione: Se N sono i bit da utilizzare e x il numero da rappresentare si utilizza il valore binario pari a

$$2^N + x$$

Es. con 4 bit

$+7 = 2^4 + 7 = 16 + 7 = 23 = \mathbf{1}0111 = 0111$

$-7 = 2^4 - 7 = 16 - 7 = 9 = 1001$

si scarta perchè abbiamo chiesto 4 bit

Conversione da decimale a complemento a 2 con segno

- Il bit più significativo (più a sx) è per rappresentare il segno (0 per il +, 1 per il -)
- Comune rappresentazione binaria per i numeri positivi
- **Regola 1:** Per i numeri negativi: inversione dei restanti bit (0→1 e 1→0) e poi si somma 1

in alternativa:

- **Regola 2:** Dati N bit, codifico in binario il numero risultato da 2^N + num
  - (es. Con 4 bit per codificare in complemento a 2 il numero -7 calcolo 16-7 = 9 e codifico 9 in binario: 1001)

Conversione da decimale a complemento a 2 con segno (Regola 1)

- -5 con quattro bit
  - il bit di segno è 1
- Conversione: $5_{10} = 0101_2$
- Inversione: 0101 → 1010
- Somma di 1: 1010 + 1 = 1011
- Verifica: + 5 → 0101
  – 5 → 1011
  = 0 = (1)0000

Applicazione del Complemento a 2: L'Addizione

- Il bit più a sinistra conserva il significato di segno.
- Il segno viene determinato automaticamente!
- Es: 15 – 5 (utilizzando 8 bit)
- Faccio la somma "normalmente"

```
 1  1111 111  (riporto)
    0000 1111 (15)
    1111 1011 (-5)
   ===========
 1  0000 1010 (10)
```

Regola – Se i primi due bit della riga dei riporti sono diversi, il risultato **non è valido**

NB: Si ignora il bit di overflow !

Conversione da complemento a 2 in decimale con segno – Regola analitica

Detto **a** un generico numero intero, la rappresentazione in complemento a due è definita secondo la seguente relazione:

$$a = a_{n-1}(-b^{n-1}) + a_{n-2}b^{n-2} + \ldots + a_2b^2 + a_1b^1 + a_0b^0$$

essendo **b** la base del sistema di numerazione (2 nel nostro caso) ed **n** il numero di cifre utilizzate per la codifica del numero

Es: $11000001 = 1 \cdot (-2^7) + 1 \cdot 2^6 + 1 \cdot 2^0 = -63$

In generale

- Con **N bit** ho 2^N configurazioni possibili
- Considerando interi positivi codifico i numeri da 0 a 2^{N-1}
- Considerando interi positivi e negativi (complemento a 2) codifico i numeri:
  - positivi: da 0 a $2^{N-1}-1$
  - negativi: da -2^{N-1} a -1

Se il numero decimale da convertire non rientra in questo range, i bit non sono sufficienti!