Università degli Studi di Cagliari
Corso di Laurea in Ingegneria Elettronica

ELEMENTI DI INFORMATICA

http://agilegroup.eu

A.A. 2015/2016

Docente: **Michele Marchesi**

CODIFICA BINARIA DELL'INFORMAZIONE

Analogico Vs. Numerico (*digitale*)

- Sistemi **analogici**: la grandezza da misurare viene rappresentata con un'altra grandezza "più pratica" da utilizzare
  - Esempio: temperatura come lunghezza colonna mercurio
- Sistemi **numerici (*digitali*):** la grandezza da misurare viene rappresentata da un numero
  - Esempi: disco in vinile Vs. CD, tachimetro analogico vs. digitale

- E' più semplice correggere gli errori nella trasmissione di numeri piuttosto che nella trasmissione di una grandezza fisica
  - Se si usa il sistema binario le cifre possibili sono solo due!

Codifica binaria

- Tutti i dati devono essere *codificati in forma binaria* per poter essere comprensibili a un calcolatore

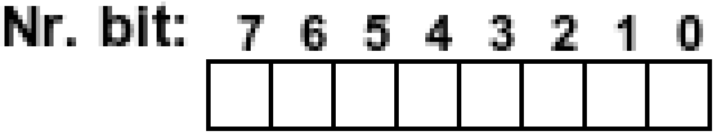
- Il *bit* è l'unità di informazione. Corrisponde allo *stato* di un dispositivo fisico a *due stati*
  – Ad es. tensione elettrica, polarizzazione di magnetizzazione

- I *due stati* vengono interpretati come 0 o 1

- Scelta di due soli stati: motivazioni tecnologiche
  – Minor probabilità di guasti ed errori

Codifica binaria (cont.)

- I bit vengono organizzati in:
  - ***byte*** (sequenze di 8 bit)



  - ***parole*** (***word***, sequenze di bit che entrano in una cella di memoria centrale. Nei calcolatori moderni sono multipli di 8, tipicamente 16, 32, 64 bit)
- I numeri interi, frazionari, i caratteri, le immagini, i suoni, ecc. possono essere tradotti in byte e parole
- Il calcolatore è in grado di operare sia la *codifica* che la *decodifica* in binario.
  - E' totalmente trasparente per l'utente e, a volte, anche per il programmatore

Codifica dei numeri naturali

- Sistema usato comunemente: *arabico*.
- Numeri rappresentati come sequenza di 10 cifre: (0,…,9)
- Sistema **posizionale**: il significato di ciascuna cifra (unità, decine, centinaia, ecc.) dipende dalla posizione che occupa nella sequenza: 123 è diverso da 321
- Altri sistemi:
  - Sistemi **additivi**: bastoncini (ciascuno rappresenta una unità), sistema di numerazione romano: I, II, III, IV, V,...

Rappresentazione posizionale dei numeri in base *p*

- Una **cifra** (digit) è un simbolo che rappresenta **nativamente** un numero da 0 a *p*-1
- Es. cifre decimali: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Un generico numero in base *p* è rappresentato dalla sequenza di **cifre** ($a_n \in \{0, ..., p-1\}$)
- $a_n, a_{n-1}, \ldots, a_0$: numero di *n+1* cifre
- Cioè il numero *N* che rappresenta è dato da:

$$N_p = a_n \cdot p^n + , \ldots , + a_0 \cdot p^0 = \sum_{i=0}^{n} a_i \cdot p^i$$

- La cifra più a sinistra è la **più significativa**, quella più a destra la **meno significativa**

Rappresentazione in base *p*

- Esempi in base decimale
(ricordiamo che $n^1 = n$; $n^0 = 1$):

  $309 = 3\ 10^2 + 0 \cdot 10^1 + 9 \cdot 10^0$

  $1234 = 1\ 10^3 + 2\ 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$

- Con k cifre in base p posso rappresentare i numeri ***naturali*** nell'intervallo: $[0, p^k - 1]$

  Es: con 3 cifre in base 10: [0, 999]
  infatti: $999 = 10^3 - 1$

Rappresentazione in base *p*

$$N_p = a_n \cdot p^n + , \ldots , + a_0 \cdot p^0 = \sum_{i=0}^{n} a_i \cdot p^i$$

- Per passare ad un'altra base *q* è sufficiente esprimere i coefficienti a_i (che sono cifre, e quindi numeri), e le potenze p^i in base *q*.
- Nei calcolatori le basi usate sono 2, 8, e 16 che corrispondono ai sistemi *binario*, *ottale*, *esadecimale*
- In caso di ambiguità, la base si può denotare come pedice (in formato decimale): 534_8, 534_{10} ,...

Il sistema binario, ottale, esadecimale

- Base p = 2. Cifre dell'alfabeto: 0 e 1.
  Es. $101001011_2 = (1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = 331_{10}$

- Base p = 8. Cifre dell'alfabeto: 0, 1, …, 7.
  Es. $534_8 = (5 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0)_{10} = 348_{10}$

- Base p = 16. Cifre dell'alfabeto: 0, 1, …, 9, A, B, C, D, E, F
  Es. $B7F_{16} = (11 \cdot 16^2 + 7 \cdot 16^1 + 15 \cdot 16^0)_{10} = 2943_{10}$

Codifiche usate in informatica

- Il sistema binario è quello fondamentale
- I sistemi ottale e esadecimale servono in pratica a rappresentare in modo sintetico i numeri binari
- Infatti, il passaggio da questi a binario e viceversa è immediato
- Base binaria: p=2; cifre $a_n \in \{0, 1\}$: bit (binary digit)
- 8 bit = 1 byte, 1024 byte = Kb, 1024^2 byte = 1Mb, Gb...
- Esempio, con n = 4:
- $1011_2 = (1 \cdot 2^3 + 0\ 2^2 + 1\ 2^1 + 1\ 2^0) = 11_{10}$
- Con n = 4 rappresento i numeri naturali nell'intervallo: [0 , 15]

Conversione da decimale a binario

- Conversione decimale in binario: **metodo delle divisioni successive**.
- Si divide il numero decimale per due fino a che il quoziente non risulta 0.
- I resti di ciascuna divisione, 0 o 1, sono le cifre binarie dalla meno significativa alla più significativa.
- **L'operazione *x mod y* (modulo) calcola il resto della divisione di *x* per *y***
- Es. 5 *mod* 2 = 1; 17 *mod* 5 = 2

Esempio di conversione da decimale a binario

- 483:2=241 (483 *mod* 2 = **1**) *bit meno significativo*
- 241:2=120 (241 *mod* 2 = **1**)
- 120:2=60 (120 *mod* 2 = **0**)
- 60:2=30 (60 *mod* 2 = **0**)
- 30:2=15 (30 *mod* 2 = **0**)
- 15:2=7 (15 *mod* 2 = **1**)
- 7:2=3 (7 *mod* 2 = **1**)
- 3:2=1 (3 *mod* 2 = **1**)
- 1:2=0 (1 *mod* 2 = **1**) *bit più significativo*

- 483_{10}=111100011_2

Spiegazione algoritmo di conversione decimale-binario

- Un numero binario a *n* bit $c_{n-1}\ c_{n-2}\ \ldots\ c_1\ c_0$ si trasforma in un intero decimale *D come*

$$D = \sum_{i=0}^{n-1} c_i \cdot 2^i$$

- Dividendo *D* per 2, dato che $D = 2q + r$
  q = quoziente, r = resto ($r = D$ mod 2)

$$r \times 2^{-1} + q = \frac{D}{2} = \sum_{i=0}^{n-1} c_i \cdot 2^{i-1} = c_0 2^{-1} + \sum_{i=1}^{n-1} c_i \cdot 2^{i-1}$$

- quindi $c_0 = r = D$ mod *2* e $\quad q = \sum_{i=1}^{n-1} c_i \cdot 2^{i-1}$

- Se q ≠ 0 (ossia, D>1) si ripete la procedura

Spiegazione algoritmo di conversione decimale-binario

- In altre parole, rappresentantando *n*:
  - se *n* è dispari, certamente la sua rappresentazione binaria termina con 1: $n \bmod 2 = 1$
  - se è pari, la sua rappresentazione binaria termina con 0 : $n \bmod 2 = 0$
  - inoltre, $q = n / 2$ (**quoziente intero**) in formato binario è rappresentato con gli stessi bit di *n,* eccetto l'ultimo:

    Es. $19_{10}=10011_2$; $(19/2)_{10}= 9_{10} = 1001_2$;
    $(9/2)_{10}= 4_{10} = 100_2$; $(4/2)_{10}= 2_{10} = 10_2$; ...

Il sistema ottale

- La numerazione ottale (2^3) è usata per “compattare” la rappresentazione di numeri binari.
- Le otto cifre ottali in formato binario:

ottale:	binario:
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Conversione da binario a ottale

- **Ottale**: si raggruppano le cifre binarie a tre a tre (a partire dalla meno significativa)
- 001010110111⇨001|010|110|111 ⇨ 1267_8
- **Spiegazione:**
- Esplicitiamo la base ed effettuiamo il raggruppamento

$$(0 \cdot 2^{11} + 0 \cdot 2^{10} + 1 \cdot 2^{9} + 0 \cdot 2^{8} + 1 \cdot 2^{7} + 0 \cdot 2^{6} + 1 \cdot 2^{5} + 1 \cdot 2^{4} + 0 \cdot 2^{3} + 1 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) = (0 \cdot 2^{2} + 0 \cdot 2^{1} + 1 \cdot 2^{0}) \cdot (2^{3})^{3} + (0 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0}) \cdot (2^{3})^{2} + (1 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0}) \cdot (2^{3})^{1} + (1 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) \cdot (2^{3})^{0} =$$

$$1 \cdot 8^{3} + 2 \cdot 8^{2} + 6 \cdot 8^{1} + 7 \cdot 8^{0}$$

Il sistema esadecimale

- La numerazione esadecimale (2^4) è usata per “compattare” la rappresentazione di numeri binari.
- Le sedici cifre esadecimali in formato binario:

decimale:	esadecimale:	binario:
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversione da binario a esadecimale

- **Esadecimale**: si raggruppano le cifre binarie a quattro a quattro (a partire dalla meno significativa)

  001010110111 ⇨ 0010|1011|0111 ⇨ $2B7_{16}$
- **Spiegazione:**
- Esplicitiamo la base ed effettuiamo il raggruppamento

  $(0 \cdot 2^{11} + 0 \cdot 2^{10} + 1 \cdot 2^{9} + 0 \cdot 2^{8} + 1 \cdot 2^{7} + 0 \cdot 2^{6} + 1 \cdot 2^{5} + 1 \cdot 2^{4} + 0 \cdot 2^{3} + 1 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) = (0 \cdot 2^{3} + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0}) \cdot (2^{4})^{2} + (1 \cdot 2^{3} + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) \cdot (2^{4})^{1} + (0 \cdot 2^{3} + 1 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) \cdot (2^{4})^{0} =$
  $2 \cdot 16^{2} + B \cdot 16^{1} + 7 \cdot 16^{0}$

Conversioni – come fare

		A			
DA	**Base**	**2**	**8**	**10**	**16**
	2		Raggruppare digit 3 a 3	Usare la definizione	Raggruppare digit 4 a 4
	8	Espandere le cifre ottali		Usare la definizione	Convertire in binario e poi in esadecimale
	10	Divisioni suc-cessive	Convertire in binario e poi in ottale (*)		Convertire in binario e poi in esadecimale (*)
	16	Espandere le cifre esadec.	Convertire in binario e poi in ottale	Usare la definizione	

(*) Si può usare anche il metodo delle divisioni successive per 8 o per 16.

Es. conversione da decimale a ottale

$N = 106_{10}$	106:2 =	53	R:	0	(converto in bin.)
	53:2 =	26		1	
	26:2 =	13		0	
	13:2 =	6		1	
	6:2 =	3		0	
	3:2 =	1		1	
	1:2 =	0		1	

$N = 1101010_2 = 001\ \ 101\ \ 010 = 152_8$

Anche:	106:8 =	13	R:	2	(divisione per 8)
	13:8 =	1		5	
	1:8 =	0		1	

$\rightarrow \quad N = 152_8$

Numero di cifre per rappresentare *n*

- Col sistema ottale, occorrono 1/3 delle cifre necessarie col sistema binario per rappresentare lo stesso numero:
  Es: $100110011011_2 = 4633_8$
- Col sistema esadecimale, occorrono 1/4 delle cifre binarie per rappresentare lo stesso numero:
  Es: $100110011011_2 = 99B_{16}$
- Col sistema decimale, un numero di *n* cifre è rappresentato in media da circa 3.3*n* cifre binarie per rappresentare lo stesso numero:
  Es: $100110011011_2 = 2459_{10}$
- Ad es., un numero dell'ordine del miliardo (10 cifre decimali) si rappresenta con circa 33 bit

Somma di due numeri

- La somma di due numeri rappresentati in base *p* si esegue come imparato in base 10, coi riporti
- Il riporto può sempre solo essere 0 (nessun riporto) o 1, qualunque sia *p*
- **Esempio:**

	decimale	binario
riporto	11	111100001111
I addendo	2381	100101001101
II addendo	1963	11110101011
somma	4344	1000011111000

Somma di delle cifre binarie

- Le cifre sono 0 e 1, il riporto può essere solo 1:

Riporto precedente	Somma	Risultato	Riporto
0	0 + 0	0	0
0	0 + 1 1 + 0	1	0
0	1 + 1	0	1
1	0 + 0	1	0
1	0 + 1 1 + 0	0	1
1	1 + 1	1	1

Somma binaria e *carry*

- Il *carry* è l'eventuale “1” che deborda se il numero di bit è insufficiente a rappresentare la somma.

  Somma esatta (nessun carry):

```
  1    ← riporto
0101 +        (5₁₀)
1001 =        (9₁₀)
------
1110          (14₁₀)
```

  Somma con carry:

```
      111    ← riporti
       1111 +        (15₁₀)
       1010 =        (10₁₀)
      -------
carry → 11001        (25₁₀ se uso 5 bit;
                      9₁₀ se  considero 4 bit: errato)
```

Numeri interi (positivi e negativi)

- Con *m* bit si possono rappresentare numeri positivi da zero a $2^m - 1$
- Rappresentazione numeri interi (anche negativi) con *m* bit:
  - **Segno e modulo**. Primo bit: il segno (0: +; 1: -)
  - **Complemento a uno**
  - **Complemento a due**

Segno e modulo

- Non posso memorizzare il “segno”, uso una codifica
- Uso il bit più a sx. per memorizzare il segno: “1” significa numero negativo, “0” numero positivo. Esempio: numeri rappresentabili con *m* bit, *m=3*:

Num. intero, base 10	Num. intero, base due, modulo e segno
–3	111
–2	110
–1	101
–0	100
+0	000
+1	001
+2	010
+3	011

Segno e modulo

- Primo bit: il segno (0: +; 1: -) Rappresento i numeri da $-(2^{m-1} - 1)$ a $(2^{m-1} - 1)$.
- Problema: doppia rappresentazione dello 0: +0, -0
- Problema: per sommare due numeri, occorre considerare i segni: se sono discordanti, occorre invece eseguire una sottrazione
- Alcuni numeri rappresentati con 8 bit:

Numero	Segno	Valore:						
12	0	0	0	0	1	1	0	0
-12	1	0	0	0	1	1	0	0
-46	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0
127	0	1	1	1	1	1	1	1
-127	1	1	1	1	1	1	1	1

Complemento a uno

- Anche in questa rappresentazione il primo bit rappresenta il segno: 0 = +, 1 = -
- I numeri positivi sono rappresentati come nel caso precedente (segno 0 seguito da numero binario)
- I numero negativi sono rappresentati scrivendo il numero negato (positivo) e ***complementando*** tutte le sue cifre (0 diventa 1, 1 diventa 0).
- Es: -10 con 5 bit: 10 è rappresentato come 01010
  complemento: 10101 -> rappresentazione di -10

Complemento a uno con 4 bit

Num. Intero, base 10	Modulo, base 2	Comple-mento a 1
-7	0111	1000
-6	0110	1001
-5	0101	1010
-4	0100	1011
-3	0011	1100
-2	0010	1101
-1	0001	1110
0	0000	1111
		0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
...	...	...

Somma di numeri in complemento a uno

- Se sono entrambi positivi, è una semplice somma binaria, di numeri positivi purché il risultato non sia un **overflow**
- Se gli addendi sono di segno diverso, e il risultato è negativo, allora il risultato è la semplice somma dei numeri come se fossero positivi
- Es: 8 – 13 con m = 5 bit:
  - 8 : 01000 , 13 : 01101, -13 : 10010
  - 01000 + 10010 = 11010 negativo
  - complemento per trovare il modulo:
    00101 = 5 – > il risultato è -5 OK!

Somma di numeri in complemento a uno

- Se gli addendi sono di segno diverso, e il risultato è positivo, oppure se sono entrambi negativi, sommandoli come se fossero numeri naturali, il risultato è quello corretto ***decrementato di uno***!
- Es: 13 - 8 con m = 5 bit:
  - 13 : 01101, 8 : 01000 , -8 : 10111
    01101 + 10111 **+ 1** = 1 | 00101 = 5
    (perdendo il "carry")
- Es: -3 -11 con m = 5 bit:
  - -3 : 11100, -11 : 10100 ,
    11100 + 10100 **+ 1** = 1 | 10001 –> 01110
    14 –> risultato è -14 OK!
    (perdendo il "carry")

Complemento a uno

- Con *m* bit posso rappresentare i numeri da $-(2^{m-1}-1)$ a $(2^{m-1}-1)$, come con modulo e segno
- Anche in questa rappresentazione lo zero è rappresentato due volte:
  - Ad es. con 5 bit: 00000 e 11111
- Le somme si possono eseguire considerando i numeri come positivi (se non c'è overflow)
- Il alcuni casi, le somme vanno "aggiustate" sommando uno al risultato
- ***Un passo avanti rispetto alla rappresentazione in segno e modulo, ma si può far meglio!***

Complemento a due

- Semplifica le operazioni aritmetiche sugli interi.
- Approccio uniforme per la somma di due numeri, qualunque sia il loro segno (e quindi anche per la differenza)
- I numeri positivi sono rappresentati come al solito: segno 0 seguito dal numero binario
- Per rappresentare -N in *m* bit, si sottrae N da 2^m. Si possono rappresentare numeri da -2^{m-1} a $(2^{m-1} -1)$
- Calcolo pratico del complemento a due di - N:
  - si complementa la rappresentazione binaria di N bit a bit
  - si somma 1 come se il risultato fosse un numero naturale.

Rappresentazione del numero intero - *N* in complemento a due a *m* + 1 bit

- Se $N = \sum_{i=0}^{m-1} a_i 2^i$, il complemento a 2 di N (*m* bit) è 2^m – N, cioè:

$$A = 2^m - \sum_{i=0}^{m-1} a_i 2^i$$

- Se uso la “regola pratica”:

$$B = \sum_{i=0}^{m-1} \overline{a}_i 2^i + 1$$

- Verifichiamo che le due rappresentazioni sono uguali, A – B = 0:

$$A - B = 2^m - \sum_{i=0}^{m-1} a_i 2^i - \sum_{i=0}^{m-1} \overline{a}_i 2^i - 1 = 2^m - \sum_{i=0}^{m-1} (a_i + \overline{a}_i) 2^i - 1 =$$

$$= 2^m - 1 - \sum_{i=0}^{m-1} 2^i = 2^m - 1 - (2^m - 1) = 0$$

Complemento a due

- Tutti i casi possibili con m = 3 bit
- Numeri interi rappresentabili: $[-2^{m-1}, 2^{m-1} - 1] = [-4, 3]$

Num. intero base 10	**Trasformazione**	**Num. intero, base 2, CPL$_2$, *m=3***
-4	8 - 4 = 4	$4_{10} = 100$
-3	8 - 3 = 5	$5_{10} = 101$
-2	8 - 2 = 6	$6_{10} = 110$
-1	8 - 1 = 7	$7_{10} = 111$
0	nessuna	$0_{10} = 000$
1	nessuna	$1_{10} = 001$
2	nessuna	$2_{10} = 010$
3	nessuna	$3_{10} = 011$

Complemento a due

- Lo zero ha una sola rappresentazione: il numero con tutti i bit posti a zero
- Tutti i numeri negativi cominciano con il bit più significativo posto a “1”, mentre tutti i positivi e lo zero iniziano con uno “0”
- Asimmetria tra negativi e positivi relativamente ai numeri rappresentabili: $[-2^{m-1}, 2^{m-1} -1]$
  - Es: se $m = 8$: $[-2^7, 2^7 -1] = [-128, 127]$

Esempi di calcolo: complemento a due

Trovare la rappresentazione compl. a 2 a 10 bit di -207_{10}

$N = 207 \rightarrow$	207:2 = 103	R:	1
	103:2 = 51		1
	51:2 = 25		1
	25:2 = 12		1
	12:2 = 6		0
	6:2 = 3		0
	3:2 = 1		1
	1:2 = 0		1

N = 0011001111

Complementato: N' = 1100110000

- Rappresentaz. di -207 = N'+1 = 1100110001

Esempi di calcolo: complemento a due

- Dato il numero compl. a 2: 10101010, trovarne la rappresentaz. decimale -N
- Osservo che m = 8 (nr. di bit) e -N è < 0 (I bit = 1)
- Sottraggo 1: N' = 10101010 – 1 = 10101001
- Complemento N' → N = 01010110
- N = 64 + 16 + 4 + 2 = 86 → $-N = -86_{10}$
- Dato il numero compl. a 2: 11100011, trovarne la rappresentaz. decimale: m = 8 (nr. di bit)
- Sottraggo 1: N' = 11100011 – 1 = 11100010
- Complemento N' → N = 00011101
- N = 16 + 8 + 4 + 1 = 29 → $-N = -29_{10}$

Somma algebrica di numeri interi rappresentati in binario

- La somma e la differenza si eseguono come in decimale
- Somma: nel caso binario si ha riporto quando il risultato > 1

Somma:

1	1			1				
0	1	1	0	0	1	1	0	+
0	0	1	1	0	1	0	1	=
1	0	0	1	1	0	1	1	

- Differenza: nel caso 0 – 1 si chiede un prestito al bit a sinistra e si esegue 10 – 1 = 1

Differenza:

	0	0				0		
0	1	1	0	0	1	1	0	-
0	0	1	1	0	1	0	1	=
0	0	1	1	0	0	0	1	

Somma e sottrazione con numeri in complemento a 2 con *m* bit

- Somme di numeri positivi e/o negativi si eseguono come se i numeri fossero senza segno (naturali)
- Sottrazione: $N_1 - N_2 = N_1 + (-N_2)$
- Cambio del segno:
  - se N > 0, complemento e sommo 1 (trascuro il carry!)
  - se N = 0 non faccio nulla
  - se N < 0, sottraggo 1 e complemento
- Overflow:
  - se sommando due numeri con segno concorde si ottiene un risultato di segno discorde, c'è **overflow** (il risultato è troppo grande per essere codificato in *m* bit)
  - se hanno segno discorde, non ci può essere overflow!

Esempi con *m* = 8 bit

45 +	00101101 +	-45 +	11010011 +
38	00100110	38	00100110
83	01010011	-7	11111001

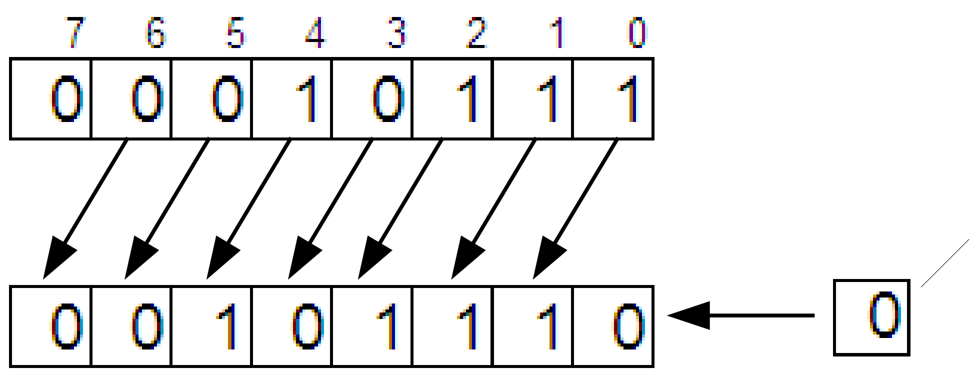
45 +	00101101 +	-45 +	11010011 +
-38	11011010	-38	11011010
7	1 00000111	-83	1 10101101

45 +	00101101 +	-45 +	11010011 +
100	01100100	-100	10011100
145	10010001	-145	1 01101111

overflow!

Operazioni di scorrimento (shift)

- Un numero intero è contenuto in una parola di memoria finita (ad es., da 8 bit)
- Sui bit di tale parola si possono eseguire operazioni di scorrimento
- Es: numero 00010111 (23_{10}) – scorrimento a sinistra:



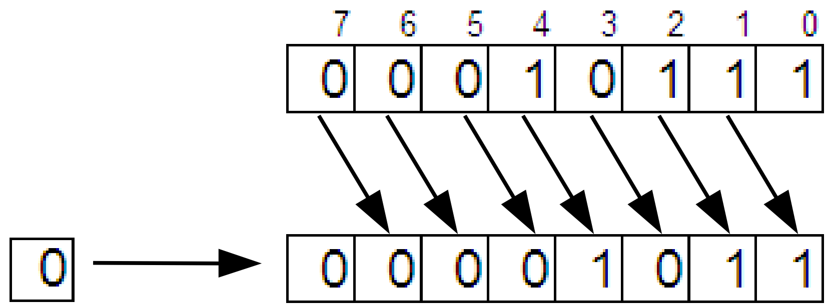
a destra è inserito 0

- Lo scorrimento a sx equivale a una moltiplicazione per 2 (salvo **overflow**): il risultato è 46

Scorrimento a destra

- Es: numero positivo 00010111 (23) – scorrimento a destra:

a sinistra
è inserito 0

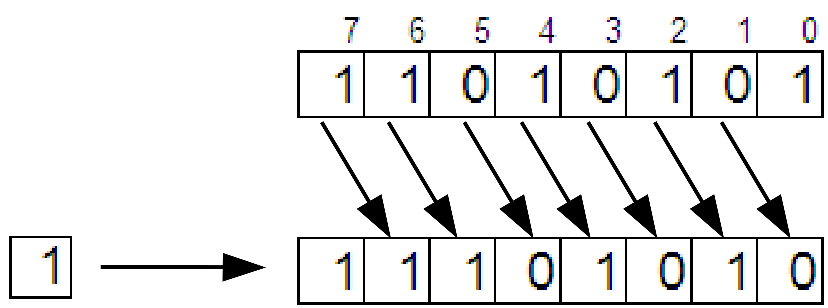


= 23

= 11

- Numero negativo complementato a due:

a sinistra
è inserito 1



= - 43

= - 22

- Lo scorrimento a dx equivale a una divisione per 2

Scorrimento a sinistra e segno

- Le moltiplicazioni / divisioni per 2 ottenute con scorrimento a sinistra / destra funzionano anche se i numeri sono negativi complementati a 2
- Sia -N il numero (N > 0). So che la sua rappresentazione con *m* bit compl. a 2 è il numero:
$$2^m - N$$
- Shift a sinistra: equivale a moltiplicare per 2:
$$(2^m - N)\,2 = 2^{m+1} - 2N = 2^m + (2^m - 2N)$$
- Ma il termine 2^m sommato equivale a un bit posto a 1 in posizione *m*, cioè il carry che viene trascurato
- Il numero restante è $2^m - 2N$, cioè la rappresentazione in compl. a 2 di -2N (salvo overflow)

Scorrimento a destra e segno

- Divisione per 2 ottenute con scorrimento a destra inserendo “1” per numeri negativi complementati a 2
- Sia -N il numero (N > 0). So che la sua rappresenta-zione con *m* bit compl. a 2 è il numero:

$$2^m - N$$

- Shift a destra inserendo 0: equivale a dividere per 2:

$$(2^m - N) / 2 = 2^{m-1} – N/2$$

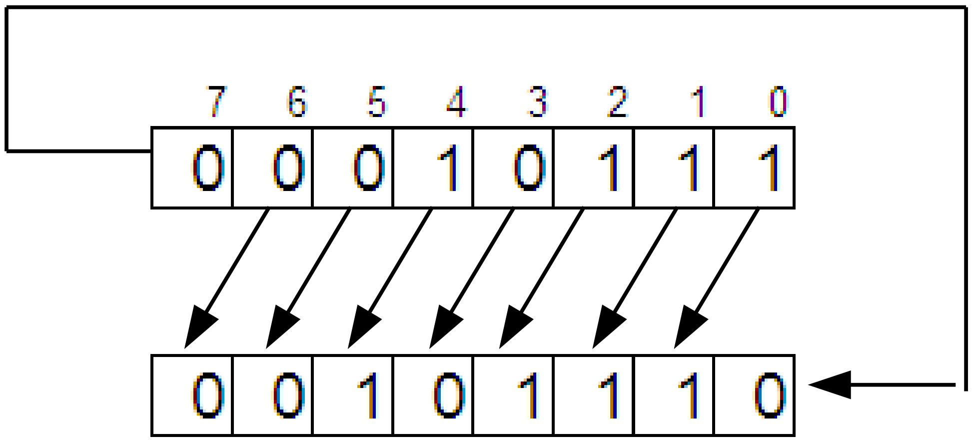
- Ma io inserisco un 1 in posizione *m* – 1 (bit più signi-ficativo). Ciò equivale a sommare 2^{m-1} al numero:

$$2^{m-1} + 2^{m-1} – N/2 = 2^m – N/2$$

- Il numero è quindi la rappresentazione in compl. a 2 di -N/2 !

Rotazione (rotate)

- Nella rotazione a sinistra, il bit che si perde a sinistra viene reinserito a destra:



- Il contrario avviene nella rotazione verso destra
- Le operazioni di rotazione non hanno un senso aritmetico, ma sono usate nella crittografia digitale

Complemento a due

- Numeri rappresentabili in 4 bit:

0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

- Somma: **6 - 2=4**:

0	1	1	0	6
1	1	1	0	-2

1

0	1	0	0	4

- Somma: **(-1)+(-4)=-5**

1	1	1	1	-1
1	1	0	0	-4

1

1	0	1	1	-5

- Shift dx: **-6 / 2 = -3**, **-7 / 2 = -4** (tronca a - ∞):

1	0	1	0	-6

1	1	0	1	-3

1	0	0	1	-7

1	1	0	0	-4

Numeri frazionari

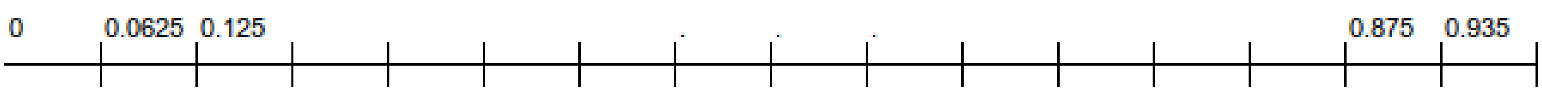
- Sono i numeri razionali fra 0 e 1: $N \in [0, 1)$
  $N = 0.a_{-1}a_{-2}\ldots a_{-n}$
- In base *p* il significato della rappresentazione è:

$$N_p = a_{-1} \cdot p^{-1} + ,\ldots, + a_{-n} \cdot p^{-n} = \sum_{i=-n}^{-1} a_i \cdot p^i$$

- Ad esempio, in base 10: 0.672 = 6/10 + 7/100 + 2/1000
- In base 2: 0.1011 = (1/2 + 0 · 1/4 + 1/8 + 1/16) = 0.6875
- Con *n* cifre in base 2, posso rappresentare numeri tra 0 e $1 - 2^{-n}$ Ad es: $n = 4$: 0.1111 = 0.9375 = 1 – 0.0625
  ma: $2^{-4} = 1 / 2^4 = 1/16 = 0.0625$

Numeri frazionari

- I numeri frazionari rappresentabili con *n* bit sono distribuiti un modo uniforme sull'asse reale (tra 0 e 1- 2^{-n}).
- Essi sono in numero di 2^{n}.
- Ad esempio, con 4 bit posso rappresentare i 16 numeri:
  .0000, .0001, .0010, .0011, … , .1100, .1101, .1110, .1111
- Tali numeri, in base decimale, sono:
  0, 0.0625, 0.125, 0.1875, … , 0.75, 0.8125, 0.875, 0.935
- Sull'asse reale:



- L'errore di approssimazione e sempre minore o uguale a $2^{-(n+1)}$

Conversione di frazionario decimale in binario (moltiplicazioni successive)

- Si moltiplica il numero per 2 e si considera la parte intera (0 o 1).
- Si continua, sottraendo dal numero l’eventuale parte intera = 1
- Ci si ferma quando si raggiunge il nr. di bit voluto, oppure se il numero residuo è 0 (improbabile)
- La successione delle parti intere è il numero binario cercato, a partire dalla cifra più significativa
- La spiegazione è analoga a quella vista per la conversione in binario di numeri interi

Moltiplicazioni successive

- 0.309 · 2 = **0.**618 - > **0** *bit più significativo*
- 0.618 · 2 = **1.**236 - > **1** - sottraggo 1
- 0.236 · 2 = **0**.472 - > **0**
- 0.472 · 2 = **0**.944 - > **0**
- 0.944 · 2 = **1**.888 - > **1** - sottraggo 1
- 0.888 · 2 = **1**.776 - > **1** - mi fermo a 6 bit

- $0.309_{10} \approx 0.010011_2$

Moltiplicazioni successive

- $0.625 \cdot 2 = \mathbf{1}.25$ - > **1** *bit più significativo*
- $0.25 \cdot 2 = 0.5$ - > **0**
- $0.5 \cdot 2 = \mathbf{1}.0$ - > **1**
- $0 \cdot 2 = \mathbf{0}$ ***FINE!***

Infatti: $0.625 = 0.5 + 0.125 = 1/2 + 1/8$

- $0.625_{10} = 0.101_2$