Notazione in complemento a 2
Algoritmo pratico

- Il procedimento funziona anche al contrario, cioè se si applica ad un numero negativo X invertendo i bit e sommando 1 si ottiene la rappresentazione del numero positivo -X.

fate voi la prova da -25 a +25

- Il procedimento funziona anche per lo zero: sommando uno a una parola di tutti uni si ottiene 0 (trascurando il riporto).

Cp2: Casi limite

- Consideriamo i casi seguenti:

-73	10110111	+73	01001001
-73	10110111	+73	01001001
------	-----------------	------	---------------
-146	(1)01101110	+146	10010010

- Il risultato è sbagliato: la prima stringa denota +110, la seconda –110.

perché secondo voi?

Notazione in complemento a 2
Overflow

- Consideriamo i casi seguenti:

-73	10110111	+73	01001001
-73	10110111	+73	01001001
------	-----------------	------	---------------
-146	(1)01101110	+146	10010010

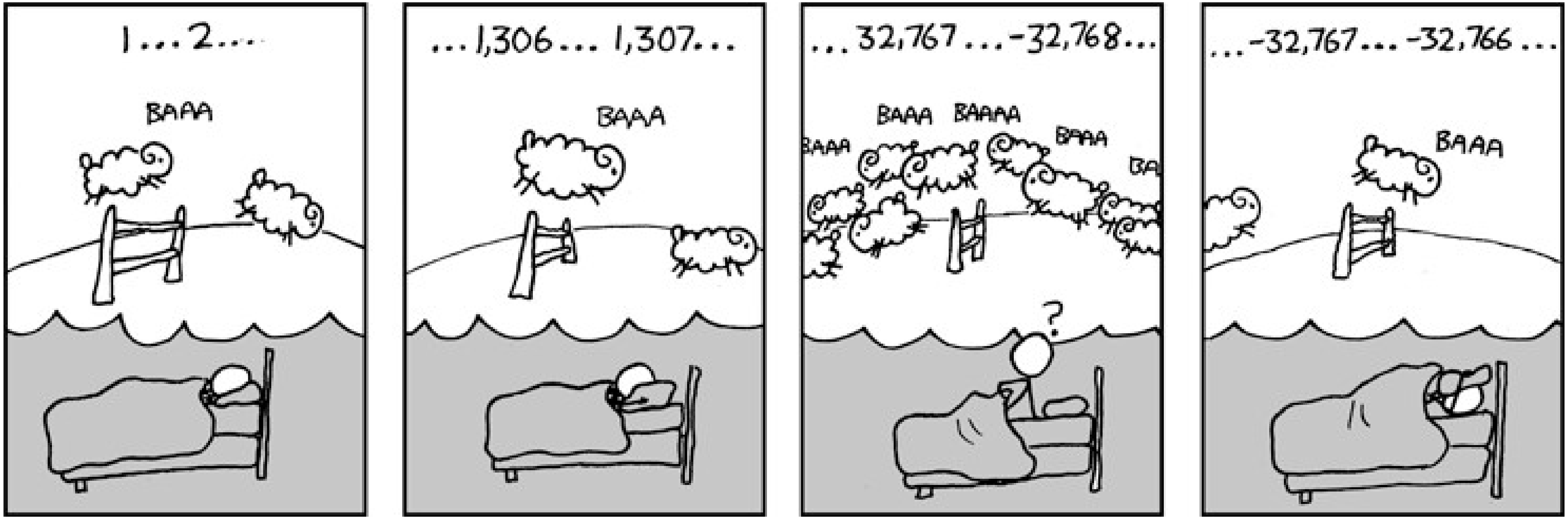
- Il risultato è sbagliato: la prima stringa denota +110, la seconda –110.
- Il motivo è che –146 e +146 sono fuori dall'intervallo rappresentabile con 8 bit (-128 ... +127).
- L'overflow si riconosce dal fatto che sommando due numeri positivi otteniamo un negativo e viceversa.

Overflow

- Sommando due valori che danno come risultato un valore esterno all'intervallo rappresentabile [$-2^{n-1} \ldots 2^{n-1}-1$] si provoca l'invasione del bit di segno da parte del risultato

- In questo caso si ha overflow: il numero di bit non è sufficiente per la rappresentazione del numero.

- Una regola pratica per vedere se si ha overflow:
  - operazioni tra numeri di segno diverso non possono provocarlo in quanto non si può uscire dal range
  - operazioni tra numeri dello stesso segno sono corrette se il risultato mantiene il segno degli addendi.


1...2...
BAAA
...1,306... 1,307...
BAAA
...32,767...-32,768...
BAAA BAAAA
BAAA
BAAA
?
...-32,767...-32,766...
BAAA

I linguaggi di programmazione

- Alcuni linguaggi come il C permettono al programmatore di stabilire:
- Su quanti bit deve essere rappresentato un intero
  - short, long, ...
- Se il numero deve essere interpretato come intero (con segno, in complemento a due) o senza segno
  - nel primo caso si avranno valori $\in$ [-2^n-1 .. $2^{n-1}-1$],
  - nel secondo $\in$ [0 .. 2^n-1].

I linguaggi di programmazione

- Rappresentazioni più usate
  - 8 bit : "`byte`" (o "`char`")
  - 16 bit (2 bytes) : "`short int`" o "`short`"
  - 32 bit (4 bytes) : "`integer`" o "`int`"
    (o "`word`", in un sistema a 32-bit)
  - 64 bit (8 bytes) : "`long int`" o "`long`"
    (o "`word`", in un sistema a 64-bit)

Di default, sono con segno (e sono in complemento a due).

Se numeri naturali: specificare "`unsigned`" ("senza segno")

es: "`unsigned char`", (8 bit senza segno)

"`unsigned int`" (32 bit senza segno)

Rappresentazione in codice eccesso B (notazione polarizzata)

- Rappresentazione impiegata nell'aritmetica a virgola mobile
- Si tratta di una rappresentazione non completamente posizionale
  - Il valore denotato da una rappresentazione su k bit $d_{k-1}\, d_{k-2} \ldots d_1\, d_0$ corrisponde a: $2^{k-1} * d_{k-1} + 2^{k-2} * d_{k-2} + \ldots + 2^1 * d_1 + 2^0 * d_0 - \mathbf{B}$
  - B corrisponde ad un bias prefissato, riferito come eccesso o polarizzazione
- Nella rappresentazione a virgola mobile IEEE 754, B assume un valore costante:
  - 127 per rappresentazioni su 8 bit (singola precisione)
  - 1023 per rappresentazioni su 11 bit (doppia precisione)
- In questa rappresentazione il numero N da rappresentare viene prima sommato ad un bias prefissato B, riferito come eccesso B (o polarizzazione B), e poi codificato come un intero privo di segno

Ad esempio, con K=8 e B=127, consideriamo la codifica 10000000: denota il valore 1 (ovvero $1*2^7-127=128-127=1$)

Rappresentazione in codice eccesso 2^{k-1}

- Un caso particolare si verifica con rappresentazioni su k bit con $B= 2^{k-1}$
- Il numero da rappresentare N viene sommato a 2^{k-1} e poi codificato come se fosse un intero privo di segno.

- Ad esempio, se k = 3 e N = -1 la rappresentazione di N in codice eccesso 2^{3-1} risulta $(-1+2^{3-1}=3) = 011$

- Per definizione, il valore di un intero v espresso in questa notazione è dato dalla formula:

$$v = - \mathbf{2^{k-1}} + 2^{k-1} * d_{k-1} + 2^{k-2} * d_{k-2} + \ldots + 2^1 * d_1 + 2^0 * d_0$$

Rappresentazione in codice eccesso 2^{k-1}

- I numeri positivi sono codificati con il MSB = 1 e il resto codificato come un numero senza segno.
- Infatti se $d_{k-1} = 1$ la formula
  - $v = - 2^{k-1} + 2^{k-1}*d_{k-1} + 2^{k-2}*d_{k-2} +... + 2^1*d_1 + 2^0*d_0$
- Si riduce a
  - $v = 2^{k-2}*d_{k-2} +... + 2^1*d_1 + 2^0*d_0$

  che è la codifica di un intero senza segno su k-1 bit.

- Esempio:
  - Con k=4 bit, la stringa 1101 denota il valore N: 4 + 1 = 5, poiché il bit 3 che ha peso +8 viene "compensato" dalla sottrazione di 2^{4-1} =8

    (dualmente: se N = 5, gli sommo 2^{4-1} = 13. La rappresentaz. di 13 in base 2: 1101)

Rappresentazione in codice eccesso 2^{k-1}

- I numeri negativi sono codificati con il MSB = 0.
- Infatti se $d_{k-1} = 0$ la formula
  - $v = - 2^{k-1} + 2^{k-1}*d_{k-1} + 2^{k-2}*d_{k-2} +... + 2^1*d_1 + 2^0*d_0$
- Si riduce a
  - $v = - 2^{k-1} + 2^{k-2}*d_{k-2} +... + 2^1*d_1 + 2^0*d_0$

poiché $2^{k-2}*d_{k-2} +... + 2^1*d_1 + 2^0*d_0 < 2^{k-1}$ $\rightarrow$ è sicuramente $v<0$

- Esempio:
  - Con k=4 bit, la stringa 0101 denota il valore -8 + 4 + 1 = -3.

Notazione in codice eccesso 2^{k-1}: proprietà

Intervallo di valori rappresentabili:

- MSB=1 $\rightarrow$ k-1 bit usabili come in binario puro $\rightarrow$ intervallo da 0 a $2^{k-1}-1$
- MSB=0 $\rightarrow$ stesso intervallo traslato di -2^{k-1} $\rightarrow$ intervallo da -2^{k-1} a -1

- Complessivamente si rappresenta l'intervallo **$[-2^{k-1} .. 2^{k-1}-1]$**

- Esempio:
  - 4 bit $\rightarrow$ 16 combinazioni $\rightarrow$ intervallo -8..7
  - Le combinazioni sono allocate metà ai positivi e metà ai negativi, come nel complemento a due.

Notazione in codice eccesso 2^{k-1} (k=3)

N	N $+2^{3-1}$	Codifica
3	(3 +4)	111
2	(2 +4)	110
1	(1 +4)	101
0	(0 +4)	100
-1	(-1 +4)	011
-2	(-2 +4)	010
-3	(-3 +4)	001
-4	(-4 +4)	000

- Il primo bit rappresenta il segno, ma a differenza del complemento a 2 i numeri negativi hanno il bit del segno a 0 ed i positivi, zero incluso, hanno il bit del segno a 1.
- Vantaggio: i numeri sono tutti ordinati in ordine crescente