INFORMAZIONI NUMERICHE

- **La rappresentazione delle *informazioni numeriche* è di particolare rilevanza**
- **Abbiamo già discusso i *numeri naturali (interi senza segno)*** N = { 0,1,2,3, …}
- **Dobbiamo discutere come rappresentare i *numeri interi (con segno)***

  $$Z = \{ -x, x \in N - \{0\}\} \cup N$$

- **.. e i *numeri reali* R, con particolare attenzione alle approssimazioni necessarie.**

NUMERI NATURALI: INTERVALLO DI VALORI RAPPRESENTABILI

- **Con N bit, si possono fare 2^N *combinazioni***
- **Si rappresentano così i numeri da 0 a 2^N-1**

Esempi

- **con 8 bit, [0 …. 255]**
  In C: unsigned char = byte
- **con 16 bit, [0 …. 65.535]**
  In C: unsigned short int (su alcuni compilatori)
  In C: unsigned int (su alcuni compilatori)
- **con 32 bit, [0 …. 4.294.967.295]**
  In C: unsigned int (su alcuni compilatori)
  In C: unsigned long int (su molti compilatori)

NUMERI INTERI (con segno)

- **Dominio: Z = { …, -2,-1,0,1,2,3, … }**
- **Rappresentare gli interi in un elaboratore pone alcune problematiche:**
  - ***come rappresentare il "segno meno"?***
  - ***possibilmente, rendere semplice l'esecuzione delle operazioni***

Magari usando gli stessi circuiti già usati per i numeri naturali…?

NUMERI INTERI (con segno)

Due possibilità:

- **rappresentazione *in modulo e segno***
  - semplice e intuitiva…
  - … ma inefficiente e complessa nella gestione delle operazioni → *non molto usata in pratica*
- **rappresentazione *in complemento a due***
  - meno intuitiva, costruita "ad hoc"
  - ma efficiente e capace di rendere semplice la gestione delle operazioni → *largamente usata*

NUMERI INTERI (con segno)

Rappresentazione *in modulo e segno*

- ***un bit per rappresentare il segno***

  0 = + 1 = -

- ***N-1 bit per rappresentare il valore assoluto***

Esempi *(su 8 bit, MSB rappresenta il segno)***:**

+ 5 = 0 0000101

- 36 = 1 0100100

NUMERI INTERI (con segno)

Rappresentazione *in modulo e segno*

Difetti:

- ***due diverse rappresentazioni per lo zero***

  + 0 = 00000000 - 0 = 10000000

- **occorrono *algoritmi speciali* per fare le operazioni**
  - **se si adottano le usuali regole, non è verificata la proprietà X + (-X) = 0**
  - **occorrono regole (e quindi circuiti) ad hoc**

NUMERI INTERI (con segno)

Rappresentazione *in m*

Difetti:

- ***due diverse rappres***

  + 0 = 00000000

- **occorrono *algoritmi s***

  operazioni
  - **se si adottano le usuali regole, non è verificata la proprietà X + (-X) = 0**
  - **occorrono regole (e quindi circuiti) ad hoc**

```
+5     0 0000101
-5     1 0000101
---    ----------
 0     1 0001010
```

Cos'è questa roba???

(+5) + (-5) = -10 ???

NUMERI INTERI (con segno)

Rappresentazione *in complemento a due*

- ***si vogliono poter usare le regole standard per fare le operazioni***
- in particolare, si vuole che
  - **X + (-X) = 0**
  - **la rappresentazione dello *zero* sia *unica***
- **anche a prezzo di una notazione più complessa, meno intuitiva, e magari *non (completamente) posizionale.***

NUMERI INTERI (con segno)

Rappresentazione *in complemento a due*

- ***idea**: cambiare il peso del bit più significativo da $+2^{N-1}$ a -2^{N-1}*
- **il peso degli altri bit rimane intoccato.**

Esempi *(su 8 bit, MSB ha peso negativo)*:

0 0000101	=	**+ 5**
1 0000101	=	**-128 + 5 = - 123**
1 1111101	=	**-128 + 125 = - 3**

NUMERI INTERI (con segno)

Rappresentazione *in complemento a due*

- ***idea**: cambi... significativ...*
- **il peso degl...**

MSB=0 → numero positivo o nullo
MSB=1 → numero negativo
Ma nel secondo caso gli altri bit non sono il valore assoluto!

Esempi *(su 8 b... MSB ha peso negativo)*:

0 0000101	=	**+ 5**
1 0000101	=	**-128 + 5 = - 123**
1 1111101	=	**-128 + 125 = - 3**

NUMERI INTERI - INTERVALLO DI VALORI RAPPRESENTABILI

Per determinare l'intervallo, osserviamo che:

- ***se MSB=0, è come per i naturali con N-1 bit***
  - **da 0 a 2^{N-1}-1** — **Esempio: su 8 bit, [0,+127]**
- ***se MSB=1, stesso intervallo traslato di -2^{N-1}***
  - **da -2^{N-1} a -1** — **Esempio: su 8 bit, [-128,-1]**
- **Intervallo globale = unione [-2^{N-1} , -2^{N-1}-1]**
  - **con 8 bit, [-128 …. +127]**
  - **con 16 bit, [-32.768 …. +32.767]**
  - **con 32 bit, [-2.147.483.648 …. +2.147.483.647]**

NUMERI INTERI - INTERVALLO DI VALORI RAPPRESENTABILI

Per determi...

- ***se MSB=0...***
  - **da 0 a $2^{N...}$**
- ***se MSB=1,...***
  - **da -2^{N-1} a -1** — **...sempio: su 8 bit, [-128,-1]**

Lo stesso intervallo
- ***prima* era tutto sui positivi [0...2^N-1]**
- ***ora* è metà sui positivi e metà sui negativi [- 2^{N-1} ... 2^{N-1}-1]**
- **lo zero rientra fra i positivi**

- **Intervallo globale = unione [-2^{N-1} , -2^{N-1}-1]**
  - **con 8 bit, [-128 …. +127]**
  - **con 16 bit, [-32.768 …. +32.767]**
  - **con 32 bit, [-2.147.483.648 …. +2.147.483.647]**

CONVERSIONE NUMERO / STRINGA

- Osservazione: poiché si opera su N bit, questa è in realtà una *aritmetica mod 2^N*
- La rappresentazione del numero *v* coincide con quella del numero $v \pm 2^N$
- In particolare, la rappresentazione del *negativo* *v* coincide con quella del *positivo* $v' = v + 2^N$

$$v = -d_{n-1}B^{n-1} + \sum_{k=0}^{n-2} d_k B^k$$

Questo è un naturale

$$v' = +d_{n-1}B^{n-1} + \sum_{k=0}^{n-2} d_k B^k$$

CONVERSIONE NUMERO / STRINGA

Esempio *(8 bit, $2^N = 256$):*

- per calcolare la rappresentazione di -3
- possiamo calcolare quella del naturale -3+256 = 253

E infatti…

- con la definizione di compl. a 2 *($2^{N-1} = 128$)*:
  -3 = -128 + 125 → "11111101"
- con il trucco sopra:
  -3 → 253 → "11111101"

CONVERSIONE NUMERO / STRINGA

Come svolgere questo calcolo *in pratica* in modo semplice?

- Osserviamo che, se v<0:

  $v = -|v|$
  $v' = v + 2^N = 2^N - |v|$
- che si può riscrivere come:

  $v' = (2^N - 1) - |v| + 1$
- dove la quantità $(2^N - 1)$ è, in binario, una ***sequenza di N "uno".***

CONVERSIONE NUMERO / STRINGA

Ma:

- se la quantità $(2^N - 1)$ è, in binario, una ***sequenza di N "uno"***,
- la sottrazione $(2^N - 1) - |v|$ si limita a ***invertire tutti i bit*** della rappresentazione di $|v|$

Infatti, ad esempio, su 8 bit:

• $2^8 - 1$ =	11111111
• se |v| =	**01110101**
• $(2^8 - 1) - \|v\|$ =	**10001010**

CONVERSIONE NUMERO / STRINGA

Conclusione:

- per calcolare il numero negativo **-|v|**, la cui rappresentazione coincide con quella del positivo **v' = (2^N -1) - |v| + 1**, occorre
- ***prima* invertire tutti i bit** della rappresentazione di **|v|** (calcolando così **(2^N -1) - |v|**)
- ***poi* aggiungere 1** al risultato

Algoritmo di complementazione a due

CONVERSIONE NUMERO / STRINGA

Esempi

- **v = -3**
  - **valore assoluto 3 → "00000011"**
  - **inversione dei bit → "11111100"**
  - **somma con 1 → "11111101"**
- **v = -37**
  - **valore assoluto 37 → "00100101"**
  - **inversione dei bit → "11011010"**
  - **somma con 1 → "11011011"**

CONVERSIONE STRINGA / NUMERO

Importante:
l'algoritmo funziona *anche a rovescio!*

- **stringa = "11111101"** → -3
  - **inversione dei bit → "00000010"**
  - **somma con 1 → "00000011"**
  - **calcolo valore assoluto → 3**
- **stringa = "11011011"** → -37
  - **inversione dei bit → "00100100"**
  - **somma con 1 → "00100101"**
  - **calcolo valore assoluto → 37**

OPERAZIONI SU NUMERI INTERI

- **La rappresentazione in complemento a due rende possibile fare *addizioni e sottrazioni con le usuali regole algebriche***
- Un primo esempio:

```
-5 +      11111011
+3 =      00000011
---       --------
-2        11111110
```

Funziona!

OPERAZIONI SU NUMERI INTERI

- **In certi casi occorre però una piccola convenzione: *ignorare il riporto***
- Un altro esempio:

```
-1 +        11111111
-5 =        11111011
---         --------
-6       (1)11111010
```

Funziona…
purché si ignori il riporto!

OPERAZIONI SU NUMERI INTERI

- **Nelle sottrazioni, analogamente, può capitare di dover *ignorare il prestito***

```
+3 -   (1)00000011 -     +3 -   (1)00000011 -
+5 =      00000101 =     -5 =      11111011 =
--        --------       --        --------
-2        11111110       +8        00001000
```

Ma.. perché ignorando prestiti e riporti funziona??

OPERAZIONI: PERCHÉ FUNZIONANO

Il motivo è semplice:

- **poiché si opera su N bit, *questa è in realtà una aritmetica modulare* di modulo 2^N**
- **ma *ignorando riporti* (o *inserendo prestiti*) si introduce *proprio* un errore pari a 2^N**
- **quindi, *mod 2^N* tale errore scompare!**

Attenzione:
possono però prodursi errori *se viene invaso il bit più significativo (bit di segno)*

ERRORI NELLE OPERAZIONI

Esempio

```
 60 +      00111100
 75 =      01100011
-----      --------
135        10011111
```

Errore!
Si è invaso il bit di segno, il risultato è *negativo*!

- **Questo errore si chiama *invasione del bit di segno* ed è una forma di *overflow***
- **Può capitare solo *sommando due numeri dello stesso segno*** (due *positivi* o due *negativi*)