Complemento al corso di Tecniche della Programmazione
Laurea in ingegneria dell'informazione

Rappresentazione di numeri

Un *numero* e' un'entità teorica, un concetto. Un numero può vivere bene nel mondo delle idee (e anche nella nostra mente).

Per usare i numeri in modo coerente bisogna però poterli rappresentare nel mondo reale (ad esempio sulla carta).

Un *numerale* è una scrittura che serve a rappresentare un numero. Si tratta di una sequenza di *cifre*. Ad esempio il numero dodici e' rappresentato usualmente dal numerale 12.

Il numerale che rappresenta un certo numero viene usato concretamente per comunicare quel numero o eseguirci dei calcoli.

In particolare, il metodo usato per rappresentare i numeri, cioè per scrivere, dato un numero, il numerale corrispondente, deve essere tale che ogni numero abbia il suo numerale e che tale numerale non rappresenti altri che quel numero.

Però sappiamo che i numeri sono infiniti e lo spazio per scrivere numerali è finito. Quindi dovremo sempre accontentarci di dire che i metodi di rappresentazione dei numeri sono tali da assegnate ad ogni "numero rappresentabile" (cioe' effettivamente esprimibile con i numerali che possiamo scrivere), il corrispondente numerale, in modo biunivoco (quel numerale rappresenta quel numero e solo quello).

Banalmente queste limitazioni di spazio per la scrittura di numerali corrispondono a limiti sulla lunghezza dei numerali stessi, cioe' a limiti sul numero di cifre utilizzabili per scrivere i numerali.

Ad esempio, se stabilissi che non posso usare numerali piu' lunghi di una cifra (numero di cifre = 1), saprei anche implicitamente che i numeri rappresentabili (secondo il metodo usuale che ci e' familiare) sono quelli da zero (0) fino a nove (9). Se invece stabilissi che non posso usare numerali piu' lunghi di due cifre (numero di cifre = 2), saprei anche implicitamente che i numeri rappresentabili (secondo il metodo usuale che ci e' familiare) sono quelli da zero (0 oppure 00) fino a novantanove (99). Se i numerali sono di 4 cifre potremo rappresentare tutti i numeri che vanno da 0000 (zero) fino a 9999 (novemilanovecentonovantanove). Solo questi 10.000 numeri sono rappresentabili con 4 "cifre decimali".

I metodi per rappresentare numeri sono tanti e diversi, e basati su concetti diversi:
ad esempio il numero dodici puo' essere rappresentato con

- 12, usando il sistema decimale che ci è familiare: un sistema posizionale (cioè tale che le cifre di un numerale hanno un valore numerico diverso a seconda di quali sono e di quale posizione occupano nel numerale) in cui le cifre usate sono 0, 1, 2, 3, 4, 5, 6,7 ,8, 9;
- XII, nel sistema additivo che usa le cifre romane (I sistemi additivi sono quelli in cui una cifra ha un valore in generale prestabilito, indipendentemente – in generale – dalla sua posizione)
- 1010, nel sistema binario (sistema posizionale con cifre binarie 0, 1) con numerali di quattro cifre;
- 001010, nel sistema binario con numerali di quattro cifre;
- 14, nel sistema ottale con numerali di due cifre);
- 000C nel sistema esadecimale (numerali di quattro cifre)

Se vogliamo usare una notazione binaria, cioe' una scrittura dei numerali usando solo le cifre 0 e 1, e ci limitiamo a tre cifre, possiamo usare i numerali 000, 001, 010, ..., 111 per rappresentare i numeri da 0 a 7:

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

Se i numerali sono di n cifre, abbiamo 2^n diversi possibili numerali (2^n diverse combinazioni di cifre); i numeri che possiamo rappresentare univocamente sono quindi 2^n e di norma sono i numeri da 0 a $(2^n - 1)$.

(Eh si', se n=3, 2^n =8 e i numeri rappresentabili sono quelli da 0 a 7 ...).

Questo tipo di rappresentazione si chiama "**in binario puro**" e rende possibile rappresentare i numeri **positivi** (solo quelli rappresentabili, cioe' appartenenti all'intervallo di rappresentazione $[0, 2^n - 1]$).

Se vogliamo rappresentare nelle locazioni di memoria i numeri **interi relativi** (cioe' positivi e negativi) le cose cambiano.

Una **prima possibilita'** e' la rappresentazione in **modulo e segno**:

Sia V il numero intero relativo; e siano n i bit (le cifre) disponibili per scrivere i numerali: il primo bit viene usato per rappresentare il segno del numero (0 positivo, 1 negativo); gli altri n-1 bit vengono usati per rappresentare il valore assoluto di V (sempre che siano abbastanza ...).

Ad esempio, con n=3, ecco i numerali (sopra) che rappresentano i numeri (sotto) da –3 a +3 in modulo e segno:

000	**0**01	**0**10	**0**11	**1**00	**1**01	**1**10	**1**11
0	1	2	3	-0	-1	-2	-3

(come viene rappresentato –2? il primo bit vale 1, perche' –2 e' negativo; i successivi bit sono quelli che rappresentano 2 in binario puro su n-1 cifre: 10)

Se i numerali sono di n bit, in generale, questa tecnica di rappresentazione consente di rappresentare i numeri interi nell'intervallo (chiuso) seguente

$$[-(2^{n-1}-1), 2^{n-1}-1]$$

eh gia', [-3, 3] nell'esempio sopra (con n=3)

Ci sono due numerali dedicati allo zero: brutto; sprechiamo un numerale per rappresentare un numero gia' rappresentato. Anche solo per una questione di estetica, questa rappresentazione non ci piace.

Inoltre questa rappresentazione rende necessario usare circuiti "piu' complicati" per eseguire le operazioni tra numeri cosi' rappresentati.

La **rappresentazione realmente usata** e' piu' simile alla seguente e si chiama **rappresentazione in complemento a due**.

Essa permette di rappresentare i numeri interi relativi senza perdere numerali e con alcune possibilita' di maggiore efficienza nei calcoli.

Non si tratta di una "combinazione tra modulo e segno e qualcosa d'altro"; si tratta di una tecnica di rappresentazione a se' stante.

L'idea di fondo e' la seguente: sia n il numero di bit con cui scriviamo i numerali e sia V il numero intero di cui vogliamo calcolare la rappresentazione in complemento a due (che chiamiamo comp(V)); allora

```
           { se V>=0       rappresentazione binaria in n bit di V
compl(V)=  {
           { se V<0        rappresentazione binaria del numero 2^n-V
```

Sia n=3, ecco la distribuzione dei numeri rappresentabili (sopra i numerali e sotto i corrispondenti numeri)

000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

I numeri rappresentabili in complemento a 2 con n bit, sono quelli nell'intervallo

$$[-2^{n-1}, 2^{n-1}-1]$$

Quindi, per n=3, da –4 a +3.

Stavolta non c'e' un numerale doppione per lo zero.

Adesso risolviamo un esercizio d'esame.

Siano **N=121, M=591**

1) Stabilire il numero minimo B tale che sia N che M possano essere rappresentati come numeri binari puri con B cifre;
2) Scrivere la rappresentazione di **N** ed **M** come numeri binari puri con B cifre ed eseguire la somma **N+M**;
3) Stabilire il numero minimo di cifre necessarie per rappresentare tanto **N** quanto **–M** in complemento a due e mostrare tali rappresentazioni, dettagliando il procedimento usato per ottenerle;
4) Infine eseguire l'operazione **N-M,** e scrivere il risultato usando 16 bit.

---

1) Stabilire il numero minimo B tale che sia N che M possano essere rappresentati come numeri binari puri con B cifre;

Risparmiando i conti, ecco la rappresentazione in binario puro dei due numeri (dato che per il primo sono sufficienti 7 bit e per il secondo ne servono almeno 10, il numero B e' 10)

$N=121_{(10)}= 1111001_{(2)}$

$M=591_{(10)}= 1001001111_{(2)}$

Dunque sono necessari 10 bit per rappresentare i due numeri in binario (B=10)

---

2) Scrivere la rappresentazione di N ed M come numeri binari puri con B cifre ed eseguire la somma N+M;

essendo il sistema binario un sietema posizionale, per scrivere il numerale 1111001 su 10 bit basta aggiungere zeri a sinistra (sono zeri; non pesano nel calcolo del numero rappresentato, che rimane lo stesso)

$N=0001111001_{(2)}$ (aggiungiamo alcuni 0 per rappresentare il numero con 10 bit)

$M=1001001111_{(2)}$

```
  1111111 ← riporto
0001111001 +
1001001111 =
-----------
1011001000
```

N+M = $1011001000_{(2)}$

facciamo la prova: 1011001000 dovrebbe essere (121+591) cioe' 712

le varie cifre hanno peso diverso, a seconda della posizione

```
cifre 1   0   1   1  0  0 1 0 0 0
peso    512 256 128 64 32 16 8 4 2 1
                    e 512+128+64+8=712
```

---

3) Stabilire il numero minimo di cifre necessarie per rappresentare tanto **N** quanto **–M** in complemento a due e mostrare tali rappresentazioni, dettagliando il procedimento usato per ottenerle;

Il minimo intervallo di numeri rappresentabili in complemento a duie, che contenga N, e'

$$[-2^{8-1}, 2^{8-1}-1]$$

quindi il numero di bit da usare per rappresentare N in complemento a due e' n=8 (7 non bastano; 9, 10, 11, ... ovviamente vanno bene lo stesso, ma non sono minimali.

Inoltre N e` positivo, quindi non faccimao altro che scriverne la rappresentazione binaria pura in 8 cifre e abbiamo il complemento a due:

$N=121_{(10)}= 01111001$

Per uanto riguarda –M, che è –591, si tratta di un numero negativo e

a. il minimo intervallo di numeri rappresentabili in complemento a due in cui –M e' contenuto e' $[-2^{11-1}, 2^{11-1}-1]$ = [-1024, 1023], quindi servono 11 bit almeno per rappresentare –M in complemento a due;
b. la sua rappresentazione in complemento a due e` il numero binario puro $2^{11}$-M = 2048 – 591 = 1457; cioe` 10110110001.

c. Pero' se non si vogliono fare i calcoli di cui al punto b., c'e` un triste ma efficace algoritmino: per ottenere il complemento a due di un numero negativo si possono invertire tutti i bit del suo valore assoluto (ottenendone il complemento a 1) e aggiungere 1: in altre parole, dovendo ottenere il complemento a due di –591,

a. stabilito che servono 11 bit,
b. 591 in 11 bit e' 01001001111
c. rivoltando i bit si ha 10110110000
d. e aggiungendo 1 si ha 10110110001

insomma

- il numero minimo di bit necessari per rappresentare in complemento a due tanto N quanto –M e' 11;
- $N = 121_{(10)} = 00001111001$
- $M = -591_{(10)} = 10110110001$

---

4) Infine eseguire l'operazione **N-M,** e scrivere il risultato usando 16 bit. Facciamo tutto usando 16 bit!

```
N = 0000000001111001
-M= 1111110110110001
```

dato che –M e` negativo, si aggiungono 1 a sinistra
(
e non puo' che essere cosi': se ad esempio 111 e` la rappresentazione in complemento a due di –1 con tre cifre, se voglio rappresentare il medesimo numero con quattro cifre, non e` che aggiungo uno 0 a sinistra, no? Certo che no! Se facessi cosi' otterrei 0111 che rappresenta 7 in complemento a due su quattro cifre; invece 1111 e` il numerale per rappresentare –1 in quattro cifre: curioso no? 111111 e' –1, 11111111 e –1, 1111111111111111 e' –1 ...
)

L'operazione N-M si fa semplicemente sommando le rappresentazioni di N e -M. Posto che non si abbia overflow, il risultato la rappresentazione in complemento a due di N-M.

```
0000000001111001
1111110110110001
----------------
1111111000101010

N-M = 1111111000101010
```

La prova che e` proprio cosi` si ottiene vedendo se questa e` proprio la rappresentazione in complemento a due su 16 bit di –470 (121-591). Ed e` lasciata ai discenti volenterosi ...