La Rappresentazione dei Numeri e le Operazioni Aritmetiche

Corso di Calcolatori Elettronici I

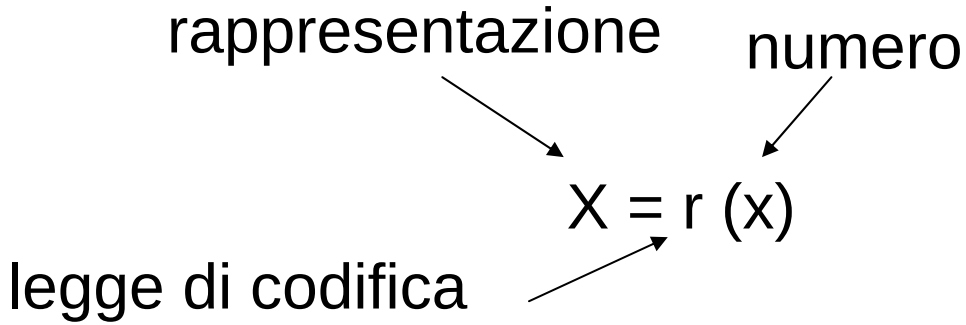
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli “Federico II”

Sommario

- Rappresentazione dei numeri (richiami)
  - Overflow e underflow
  - La rappresentazione di numeri naturali
  - La rappresentazione di numeri relativi
  - Rappresentazione di numeri reali
- Operazioni aritmetiche
  - Segno e modulo
  - Complementi alla base
  - Complementi diminuiti
  - Eccesso-k

Introduzione

- Pressoché tutte le applicazioni, per realizzare le loro elaborazioni, utilizzano calcoli numerici, indici, contatori, etc…
- Così come per qualsiasi altro tipo di dato, anche i numeri, per essere immagazzinati nella memoria di un calcolatore, devono essere tradotti in sequenze di bit
- Questa operazione è a tutti gli effetti analoga ad una generica operazione di codifica
- In particolare, l'alfabeto sorgente è costituito da numeri



Strategie di codifica

- Tipicamente, si adotta una codifica a lunghezza fissa
- Il numero di bit varia a seconda della cardinalità dell'insieme dei numeri che si desidera rappresentare
  - Nella pratica, resta comunque pari ad un multiplo di 8 bit (tipicamente 8, 16, 32, 64 bit).
- L'associazione di un numero alla parola codice viene
  - Realizzata differentemente a seconda della tipologia di numeri che si desidera rappresentare
    - (naturali, relativi, razionali, etc…).
  - Influenzata da aspetti che mirano a preservare la facile manipolazione delle rappresentazioni da parte del calcolatore
    - (operazioni aritmetiche, confronti logici, etc…).

Overflow - Definizione

- Sia la dimensione che il numero dei registri in un calcolatore sono finiti
- La cardinalità degli insiemi numerici che si rappresentano è, invece, infinita
- È inevitabile dunque che in un insieme di cardinalità infinita solo un sotto-insieme finito di elementi possa essere rappresentato
- Gli operatori aritmetici, pur essendo talvolta chiusi rispetto all'intero insieme, quasi certamente non lo sono rispetto al sotto-insieme di cardinalità finita
- Quando accade che, per effetto di operazioni, si tenta di rappresentare un numero non contenuto nel sotto-insieme si parla di *overflow*

Overflow - Esempio

- Si assuma di rappresentare i numeri naturali mediante un'operazione di codifica sul sottoinsieme da 0 a 127 (7 bit)
  - La somma 100 +100 genera un overflow, essendo il numero 200 non rappresentabile nel sottoinsieme
  - La differenza 5 – 10 genera un overflow essendo il numero –5 non rappresentabile nel sottoinsieme

Errore di approssimazione

- Nel caso della rappresentazione di numeri razionali sussiste il problema dell'overflow, ma subentra anche il problema dovuto alla caratteristica dei numeri razionali di costituire un insieme *denso*.
- A causa di ciò, anche se si sceglie per la rappresentazione un intervallo limitato, non tutti i numeri all'interno di esso potranno essere rappresentati.
- Quando si tenta di rappresentare un numero per cui non è stata prevista una rappresentazione, spesso si associa a tale numero la rappresentazione del numero "più vicino", cioè quello che lo *approssima* meglio
- In questo caso si parla di *errore di approssimazione*

Underflow

- Quando in un'elaborazione si tenta di rappresentare un numero "troppo vicino" allo zero, l'errore di approssimazione può far sì che la rappresentazione scelta sia quella dello zero
- Questa evenienza può condizionare pesantemente i calcoli successivi nel seguito dell'elaborazione a causa delle peculiarità della cifra *zero* (che, ad esempio, non può essere utilizzata al denominatore di una frazione)
- Si parla allora di *underflow*.
- Questa problematica è molto sentita nell'ambito del calcolo numerico, specialmente nelle applicazioni che, per loro natura, tendono a lavorare con quantità molto piccole (p.es. applicazioni di calcolo differenziale)

Rappresentazione dei numeri naturali

- Si voglia rappresentare attraverso sequenze di bit di lunghezza costante pari ad n l'insieme dei numeri naturali
  - Il numero degli elementi rappresentabili è pari a 2^n
  - Tipicamente, volendo rappresentare sempre anche lo zero, si rappresentano i numeri compresi tra 0 e $2^n - 1$
- L'associazione tra ogni numero e la propria rappresentazione avviene, nei casi pratici, nella maniera più intuitiva
  - Ad ogni numero viene associata la stringa di bit che lo rappresenta in un sistema di numerazione binario posizionale.
- L'overflow avviene quando si tenta di rappresentare un numero esterno all'intervallo $[0; 2^n - 1]$

Esempio

- Rappresentazione dei numeri naturali su 4 bit
  - n=4
  - Intervallo: [0;15]
  - Codifica: X=x

x	X_2	X_{10}
15	1111	15
14	1110	14
13	1101	13
12	1100	12
11	1011	11
10	1010	10
9	1001	9
8	1000	8
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0

Operazioni sui numeri naturali su 4 bit

- Per realizzare le operazioni, il calcolatore può lavorare direttamente sulle rappresentazioni
- La correttezza dei calcoli è garantita dalle leggi dell'aritmetica binaria posizionale (analoghe a quelle della classica aritmetica decimale)
- L'overflow può essere facilmente rilevato attraverso la valutazione del riporto (o del prestito) sull'ultima cifra.

Esempi

6+ 8= ------- 14	0110+ 1000= ------- 1110	11 – 5= ------- 6	1011– 0101= ------- 0110
14+ 3= ------- 17	1110+ 0011= ------- overflow 10001	9– 7= ------- 2	1001– 0111= ------- 0010

0101× 0011= ---------- 0101 0101= 0000== 0000=== ---------- 0001111

Rappresentazione in segno e modulo – 1/2

- Mentre nel caso dei numeri naturali l'associazione di un numero con una stringa di bit è alquanto intuitiva, nel caso dei numeri relativi si pone il problema di associare una rappresentazione a ciascun numero negativo.
- Una possibilità è:
  - Associare un singolo bit (per esempio quello più significativo) al segno
  - Utilizzare i restanti bit per rappresentare il modulo (che è un numero naturale).
- Questo tipo di codifica va sotto il nome di *"codifica in segno e modulo"*

Rappresentazione in segno e modulo – 2/2

- Se si utilizzano n bit:
  - La legge di codifica X=r(x) è: $X = |x| + 2^{n-1} * \text{sign}(x)$
  - Si possono rappresentare i numeri relativi compresi nell'intervallo $[-(2^{n-1} - 1); 2^{n-1} - 1]$
  - In totale i numeri rappresentati sono $2^n - 1$ (e non 2^n)
    - Ciò dipende dal fatto che, a causa della natura di questa codifica, lo zero ammette due possibili rappresentazioni dette *zero positivo* e *zero negativo*.

Esempio

- Rrappresentazione in segno e modulo su 4 bit
  - n=4
  - Intervallo: [-7;7]
  - Codifica:
    - X=|x|+8 * sign(x)

x	X_2	X_{10}
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000;1000	0;8
-1	1001	9
-2	1010	10
-3	1011	11
-4	1100	12
-5	1101	13
-6	1110	14
-7	1111	15

Operazioni in segno e modulo su 4 bit

- Diversamente dalla rappresentazione dei numeri naturali, questa volta non è possibile lavorare direttamente sulle rappresentazioni dei numeri per realizzare le operazioni aritmetiche. È necessario, invece, lavorare separatamente sul segno e sul modulo.
- Quando, ad esempio, si sommano due numeri di segno discorde, bisogna determinare quello con modulo maggiore e sottrarre ad esso il modulo dell'altro. Il segno del risultato sarà quello dell'addendo maggiore in modulo.
- Tale caratteristica, insieme con il problema della doppia rappresentazione dello zero, rende i calcoli particolarmente laboriosi e, per questo motivo, non è molto utilizzata nella pratica.

Rappresentazione in complementi alla base

- Una seconda tecnica per la rappresentazione dei numeri relativi consiste nell'associare a ciascun numero il suo resto modulo $M=2^n$, definito come:

$$|x|_M = x-[x/M]*M$$

- Questo tipo di codifica, su n bit, è equivalente ad associare:
  - il numero stesso (cioè X=x), ai numeri positivi compresi tra 0 e $2^{n-1} – 1$;
  - il numero $X = 2^n – |x|$, ai numeri negativi compresi tra 2^{n-1} e – 1;
- I numeri rappresentati sono quelli compresi nell'intervallo

$$[–2^{n-1}; 2^{n-1} – 1]$$

Esempio

- Rappresentazione in complementi alla base su 4 bit
  - n=4
  - Intervallo: [– 8;7]
  - Codifica:
    - X=x; 0<=x<=7
    - X= 2n – |x|; 8<=x<= –1

x	X_2	X_{10}
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0
-1	1111	15
-2	1110	14
-3	1101	13
-4	1100	12
-5	1011	11
-6	1010	10
-7	1001	9
-8	1000	8

Complementi alla base: proprietà

- Questa rappresentazione ha il fondamentale vantaggio di permettere, nell'ambito di operazioni aritmetiche, di lavorare direttamente sulle rappresentazioni.
- La regola sulla quale questa affermazione si basa è la seguente:

  la rappresentazione della somma (algebrica) di x ed y si ottiene come somma (modulo-M) delle rappresentazioni di x e y; analoghe sono le proprietà della differenza e del prodotto.

- Questo tipo di codifica conserva, inoltre, la proprietà delle rappresentazioni di avere il primo bit alto se (e solo se) il corrispondente numero è negativo.

Esempi di addizioni in complementi alla base

2+		0010+		– 2+		1110+	
– 6=	→	1010=		– 3=	→	1101=	
----------		----------		----------		----------	
–4		1100		–5		11011	

si ignora

somma modulo-16

È possibile effettuare la somma direttamente tra le rappresentazioni modulo-M: il risultato ottenuto in questo modo, è proprio la rappresentazione (modulo-M) del risultato corretto.

Complementi alla base: la complementazione

- In complementi alla base, a partire dalla rappresentazione di un numero, è anche particolarmente semplice ottenere la rappresentazione del suo opposto
- È infatti sufficiente *complementare tutti i bit a partire da sinistra, tranne l'uno più a destra ed eventuali zero successivi*.
- Questa ulteriore caratteristica consente di realizzare le sottrazioni attraverso la composizione di una complementazione (nel senso suddetto) ed un'addizione.
- Nell'aritmetica in complementi alla base, di conseguenza, l'addizionatore e il complementatore rappresentano i componenti fondamentali per la realizzazione di tutte le operazioni.

Esempi di complementazione su 4 bit

- La rappresentazione di 6_{10} su 4 bit è 0110_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1010_2.
- 1010_2 è la rappresentazione di –6 in complementi alla base.

- La rappresentazione di 5_{10} su 4 bit è 0101_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1011_2.
- 1011_2 è la rappresentazione di –5 in complementi alla base.

- La rappresentazione di 1_{10} su 4 bit è 0001_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1111_2.
- 1111_2 è la rappresentazione di –1 in complementi alla base.

Esempio di moltiplicazione in complementi alla base

```
   2*          0010×
  – 3=         1101=
-----------  ------------
   – 6          0010
                0000=
               0010==
              0010===
             ------------
             00011010
```

si ignora

prodotto

Complementi diminuiti

- La rappresentazione in complementi diminuiti costituisce un'ulteriore alternativa per la codifica dei numeri relativi.
- Concettualmente è analoga alla rappresentazione in complementi alla base.
- La differenza rispetto ad essa è che la legge di codifica dei numeri negativi è leggermente differente:
  - $X=2^n - |x|;$ (complementi alla base)
  - $X=2^n - 1 - |x|;$ (complementi diminuiti)
- I numeri rappresentabili, se si utilizzano n bit, sono quelli compresi nell'intervallo $[-(2^{n-1} - 1); 2^{n-1} - 1]$.
- Anche in questo caso, quindi, le cifre rappresentabili sono $2^n - 1$ e ancora una volta è lo zero ad avere una doppia rappresentazione.

Esempio

Rappresentazione in complementi diminuiti su 4 bit

n=4

Intervallo: [– 7;7]

Codifica:

$X=x; \quad 0<=x<=7$

$X= 2^n – 1 – |x|; \quad – 7<=x<= – 1$

x	X_2	X_{10}
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000;1111	0;15
-1	1110	14
-2	1101	13
-3	1100	12
-4	1011	11
-5	1010	10
-6	1001	9
-7	1000	8

Esempio

- Rappresentazione in complementi diminuiti su 4 bit
  - n=4
  - Intervallo: [– 7;7]
  - Codifica
    - X=x; 0<=x<=7
    - X= 2n – 1 – |x|; –7<=x<=–1

x	X_2	X_{10}
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000;1111	0;15
-1	1110	14
-2	1101	13
-3	1100	12
-4	1011	11
-5	1010	10
-6	1001	9
-7	1000	8

Complementi diminuiti: perché?

- Maggiore semplicità con cui è possibile calcolare la rappresentazione dell'opposto di un numero, a partire dalla rappresentazione del numero stesso: basta semplicemente complementare tutti i bit della rappresentazione indistintamente.
- Esempio:
  - la rappresentazione in complementi diminuiti su 4 bit di 4 è 0100;
  - complementando tutti i bit si ottiene 1011;
  - 1011 è la rappresentazione in complementi diminuiti su 4 bit di –4.

  - la rappresentazione in complementi diminuiti su 4 bit di – 6 è 1001;
  - complementando tutti i bit si ottiene 0110;
  - 0110 è la rappresentazione in complementi diminuiti su 4 bit di 6.

Aritmetica in complementi diminuiti

- Componenti:
  - Ancora l'addizionatore modulo-2^n (e non 2^{n-1})
    - L'addizionatore modulo-2^n è più semplice da realizzare
  - Un complementatore.
- Il risultato però deve essere opportunamente "corretto" per renderlo compatibile con l'aritmetica in modulo 2^{n-1}.
- In particolare deve essere aggiunta un'unità al risultato nei seguenti casi:
  - se entrambi gli addendi sono negativi;
  - se un addendo è positivo, l'altro negativo e la somma è positiva.
- Nei casi suddetti l'aritmetica degli interi positivi darebbe overflow
  - L'overflow quindi può essere interpretato come la necessità di effettuare la correzione

Esempi di somme in complementi diminuiti.

– 2+		1101+
– 3=		1100=
---	→	---
– 5		**1**1001+
	overflow	1=

		1010

Somma di due numeri negativi.
Si è generato overflow nell'aritmetica degli interi positivi.
Necessita correzione.

5+		0101+
– 2=		1101=
---	→	---
3		**1**0010+
	overflow	1=

		0011

Somma di un numero positivo e un numero negativo.
Il risultato è positivo.
Si è generato overflow nell'aritmetica degli interi positivi
Necessita correzione.

3+		0011+
– 4=		1011=
---	→	---
– 1		1110

Somma di un numero positivo e un numero negativo.
Il risultato è negativo.
Non si è generato overflow nell'aritmetica degli interi positivi
Non necessita alcuna correzione.

Rappresentazione eccesso-k

- La rappresentazione in eccesso-k costituisce un metodo diverso da quello dei resti in modulo per ricondurre i numeri negativi a positivi.
- In particolare, tutti i numeri sono traslati verso l'alto di k, che viene scelto maggiore o uguale al numero più piccolo da rappresentare: $X = r(x) = x+k$

Rappresentazione eccesso-k - Proprietà

- Analogamente al caso dei complementi diminuiti, la somma va corretta aggiungendo o sottraendo la costante k, e quindi in maniera sufficientemente semplice
- Le moltiplicazioni e le divisioni risultano invece più complesse.
- Il vantaggio di tale codifica è che viene conservata la proprietà della disuguaglianza sulle rappresentazioni:

$$x_1 > x_2 \Leftrightarrow X_1 > X_2$$

- Questa rappresentazione, perciò, è utilizzata soltanto laddove siano richieste fondamentalmente somme algebriche e confronti logici fra gli operandi.
- Tipicamente si utilizza per rappresentare gli esponenti nella rappresentazione in virgola mobile

Esempio

- Rappresentazione in eccesso-8 su 4 bit
  - n=4
  - Intervallo: [–8;7]
  - Codifica:
    - X=x+k;

x	X_2	X_{10}
7	1111	15
6	1110	14
5	1101	13
4	1100	12
3	1011	11
2	1010	10
1	1001	9
0	1000	8
-1	0111	7
-2	0110	6
-3	0101	5
-4	0100	4
-5	0011	3
-6	0010	2
-7	0001	1
-8	0000	0

Numeri reali in virgola fissa

- Quando di un numero frazionario si rappresentano separatamente la parte intera e la parte frazionaria si parla di rappresentazione in *virgola fissa*.
- La rappresentazione dei due contributi (che sono numeri interi) può essere realizzata secondo una delle tecniche viste in precedenza.
- In questo caso la posizione della virgola è fissa e resta sottintesa.

Numeri reali in virgola mobile

- Un numero reale x può essere rappresentato dalla coppia

$$(m,e)$$

tale che:

$$x = m*b^e$$

dove:

  - *m* è detta *mantissa*;
  - *e* è detto *esponente*;
  - *b* è la base di numerazione adottata.

- Il metodo, in questo caso, prende il nome di *codifica in virgola mobile*.

Normalizzazione

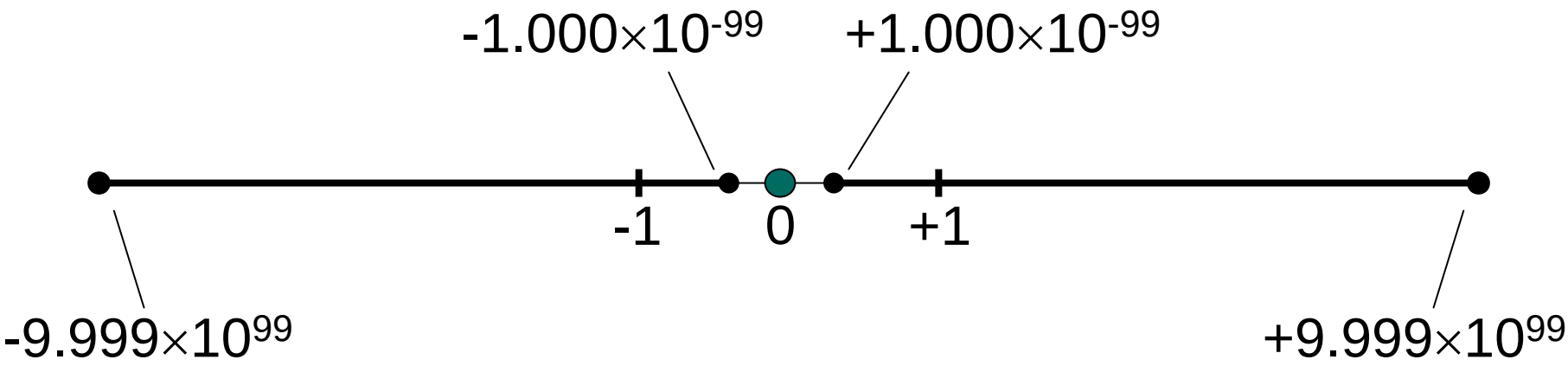
- Per ciascun numero esistono infinite coppie che lo rappresentano.
- Esempio (b=10):
  - 346.09801 è rappresentato da
    - (346.09801, 0) oppure
    - (346098.01, -3) oppure
    - (0.034609801, 4) etc…
- Allo scopo di uniformare le rappresentazioni, si fissa convenzionalmente la posizione della virgola subito dopo la prima cifra significativa, ottenendo l'*unico rappresentante*:

(3.4609801, 2)

Esempio: intervallo di rappresentazione

- Con b=10, usando 4 cifre per *m* e 2 per *e* (più due bit per i relativi segni), l'insieme rappresentabile (utilizzando solo rappresentazioni normalizzate) è:

$$[-9.999\times10^{99},-1.000\times10^{-99}]\cup\{0\}\cup$$
$$\cup[+1.000\times10^{-99},+9.999\times10^{99}]$$



Approssimazione

- Come è facile verificare, in questo tipo di rappresentazione l'approssimazione non è costante.
- In particolare la precisione assoluta è molto spinta in prossimità dello zero e va diminuendo progressivamente a mano a mano che il numero aumenta (in valore assoluto).
- Ad esempio:
  - in prossimità dello zero l'errore massimo che può essere commesso è pari a $1.001*10^{-99}$- $1.000*10^{-99}$= $0.001*10^{-99}$;
  - in prossimità dell'estremo superiore dell'intervallo di rappresentazione, invece, l'errore massimo che si può commettere è $9.999*10^{99}$-$9.998*10^{99}$=$0.001*10^{99}$.
- Si commettono quindi "errori piccoli" su "numeri piccoli" ed "errori grandi" su "numeri grandi".
- Quello che resta inalterato è invece l'errore relativo, costante su tutto l'asse di rappresentabilità.

Overflow e Underflow

- L'errore relativo dipende dal numero di cifre della mantissa.
- Gli estremi dell'intervallo di rappresentazione dipendono dal numero di cifre dell'esponente.

- Nel caso precedente di 2 cifre per l'esponente, si ha overflow per numeri maggiori (in modulo) di 10^{99} e si ha underflow per numeri minori (in modulo) di 10^{-99}.