Rappresentazione dell' informazione

Cenni, necessari per capire le caratteristiche dei tipi di dato e delle limitazioni dell'aritmetica del computer

Cos'è l'informazione ?

Tutto quello che viene manipolato da un calcolatore:

- Numeri (naturali, interi, reali, . . .)
- Caratteri ('a', 'b',...'0',...,'9','%',.....)
- Stringhe ("aba","gatto"....)
- Immagini
- Suoni
- Programmi
- .....

Cos'è l'informazione ?

Tutta l'informazione presente sul computer deve essere memorizzata in forma binaria, come sequenze di 0 ed 1:

- Le unità di memorizzazione sono formate da aggregati di dispositivi fisici elementari con due stati, uno associato a 0 e l'altro associato ad 1
- È molto più semplice da realizzare rispetto a qualcosa che distingue fra tre o più stati

Rappresentazione dei naturali

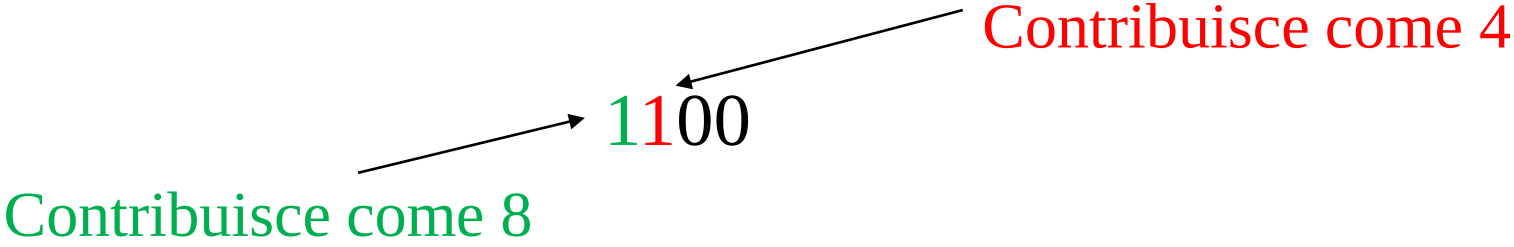
- Un numero naturale e un oggetto matematico, che puo essere rappresentato mediante una sequenza di simboli di un alfabeto fissato
- Il numero 12 puo' essere rappresentato in molti modi:
  - **XII** numerazione romana (additiva)
  - **12** sequenza dei due caratteri in rappresentazione decimale posizionale
  - **1100** rappresentazione binaria posizionale
  - **dodici** rappresentazione come stringa di caratteri
  - .....

Rappresentazione dei naturali

- Siamo interessati alle rappresentazioni in cui sia facile manipolare i naturali con le comuni operazioni aritmetiche
- Quindi entreremo in dettaglio sulle caratteristiche della rappresentazione posizionale nelle varie basi
- In queste rappresentazioni una cifra contribuisce con un valore diverso al numero a seconda della posizione in cui si trova, es:

Rappresentazione dei naturali

- Siamo interessati alle rappresentazioni in cui sia facile manipolare i naturali con le comuni operazioni aritmetiche
- Quindi entreremo in dettaglio sulle caratteristiche della rappresentazione posizionale nelle varie basi
- In queste rappresentazioni una cifra contribuisce con un valore diverso al numero a seconda della posizione in cui si trova, es:



Rappresentazione posizionale

- Un numero è rappresentato da una sequenza finita di cifre di un certo alfabeto:

$$c_{n-1}c_{n-2}\ldots\quad c_1c_0 = N_b$$

  - c_{n-1} è detta cifra più significativa
  - c_0 è detta cifra meno significativa

- Il numero *b* di cifre diverse (dimensione dell'alfabeto) e detto base del sistema di numerazione posizionale

Rappresentazione posizionale in base *b*

- Ad ogni cifra è associato un valore compreso tra *0* e *b-1* ad esempio

Base	Alfabeto	Sistema
2	01	Binario
8	01234567	Ottale
10	0123456789	Decimale
16	0123456789ABCDEF	Esadecimale

Rappresentazione posizionale in base b

$$c_{n-1}c_{n-2}\cdots \quad c_1c_0 = N_b$$

Il numero N rappresentato da una sequenza di cifre N_b dipende dalla base b

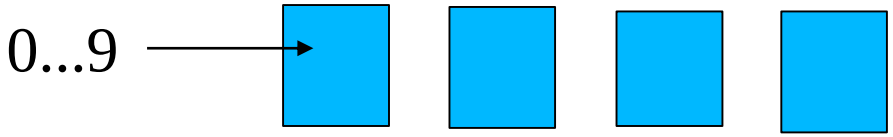
$$b^{n-1} * c_{n-1} + \cdots + b^0 * c_0 = \sum_{i=0}^{n-1} b^i * c_i = N$$

La stessa rappresentazione ha valori diversi in basi diverse

Sistema	Base	Rappresentazione	Valore (base 10)
Binario	2	101	5
Ottale	8	101	65
Decimale	10	101	101
Esadecimale	16	101	257

Intervallo di rappresentazione

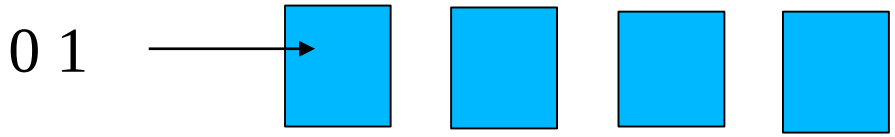
- Fissato il numero di cifre e una base abbiamo solo un numero finito di rappresentazioni
  - Sono tutte le combinazioni possibili di cifre
  - Ad esempio con 4 posizioni e rappresentazione decimale



  - si hanno le combinazioni da 0 a 9999
  - Cioè 10000= $10x10x10x10 = 10^4$ combinazioni nell'intervallo 0 ...10^4-1
  - Questo viene detto intervallo di rappresentazione

Intervallo di rappresentazione

- Vediamo un esempio con la numerazione binaria:
  - Ad esempio con 4 posizioni cifre 0, 1



  - si hanno le combinazioni da 0 a 1111
  - Cioè $16 = 2x2x2x2 = 2^4$ combinazioni nell'intervallo di rappresentazione 0 ...2^4-1
  - Questo vale in generale, in base b ed n posizioni si ha la possibilità di rappresentare al più b^n numeri diversi con intervallo di rappresentazione 0...b^n-1

Conversioni da base 10 a base b

$$b^{n-1} * c_{n-1} + \cdots + b^1 * c_1 + b^0 * c_0 = N$$

Basta osservare che

- c_0 è il resto della divisione di N per b
- $b^{n-2} * c_{n-1} + \cdots + b^0 * c_1$ è invece il quoziente della divisione per b
- Quindi effettuado delle divisioni successive per b possiamo ottenere le cifre della rappresentazione in base b (dalla mano significativa alla più significativa)

Conversioni da base 10 a base *b*

```
/* num numero da convertire
   b base */
i = 0;
while (num ! = 0) {
   c[i] = num % b;
   num = num / b;
   i++;
}
```

- Dove abbiamo rappresentato con **`c[i]`** l'i-esima cifra c_i della rappresentazione
- *Si tratta di array, aggregazioni di variabili omogenee, ne parleremo nelle prossime lezioni*

Conversioni da base 10 a base *b*

```
/* num numero da convertire
   b base */
i = 0;
while (num ! = 0) {
   c[i] = num % b;
   num = num / b;
   i++;
}
```

- Lo implementeremo in laboratorio....

Rappresentazione dei numeri nel computer

- Supponiamo ora di avere a disposizione una parola di *n* bit per rappresentare un numero
- Discutiamo come rappresentare, naturali, interi e reali
- Abbiamo come intervallo di rappresentazione

  $0 \ldots 2^n - 1$

In totale, abbiamo 2^n codifiche e dobbiamo decidere come usarle.....

Rappresentazione dei numeri naturali

- Se abbiamo a disposizione n bit
- Abbiamo come intervallo di rappresentazione di $0 \ldots 2^n - 1$
- Possiamo quindi rappresentare questo intervallo dei naturali associando ad ogni rappresentazione il numero corrispondente
- È quello che fanno i tipi **`unsigned`** del C

Rappresentazione dei numeri naturali

- Prime considerazioni .....
  - In ogni computer, abbiamo solo un insieme finito di naturali (si usano parole di lunghezza finita)
  - L'insieme non è più chiuso rispetto agli usuali operatori aritmetici !
    - Esempio: su 4 cifre decimali 9999+1=10000 non più rappresentabile ...
    - Avviene ha un fenomeno detto overflow
    - Bisogna decidere cosa fare ....
      - Ragionare in modulo, segnalare una eccezione, dipende dal linguaggio usato ....

Rappresentazione dei numeri naturali

- .. ulteriori considerazioni .....
  - Le operazioni aritmetiche non sono più associative e commutative ... Ad esempio:
    - (9999+1)-9000 da overflow mentre
    - 9999(+1-9000) da 1000 che può essere rappresentato correttamente
    - L'ordine in cui si applicano gli operatori diventa rilevante per la correttezza del calcolo!

Rappresentazione dei numeri interi

- Abbiamo a disposizione n bit, con intervallo di rappresentazione $0 \ldots 2^n-1$
- Dobbiamo dividerlo fra interi negativi e positivi!
- Considereremo due strategie
  - Rappresentazione in modulo e segno
  - Rappresentazione in complemento alla base (a 2 nel nostro caso)

Interi: modulo e segno

- Usiamo il bit più significativo per rappresentare il segno (0 positivo 1 negativo) ed il resto dei bit per il modulo,
- quindi per il modulo abbiamo a disposizione *n-1* bit, con intervallo di rappresentazione 0 ...2^{n-1}-1
- Esempio con 4 cifre binarie abbiamo

binario	decimale segno	binario	decimale segno	binario	decimale segno	binario	decimale segno
0000	+0	0100	+5	1000	-0	1100	-4
0001	+1	0101	+6	1001	-1	1101	-5
0010	+2	0110	+7	1010	-2	1110	-6
0011	+3	0111	+8	1011	-3	1111	-7

Interi: modulo e segno

Problemi :

- Doppia rappresentazione dello 0
- Gli algoritmi per le operazioni aritmetiche sono complicati (analisi per casi ...)
- Entrambe le cose sono risolte dalla rappresentazone in complemento

binario	decimale segno	binario	decimale segno	binario	decimale segno	binario	decimale segno
0000	+0	0100	+5	1000	-0	1100	-4
0001	+1	0101	+6	1001	-1	1101	-5
0010	+2	0110	+7	1010	-2	1110	-6
0011	+3	0111	+8	1011	-3	1111	-7

Interi: complemento alla base

Un intero x viene rappresentato (b base n cifre) da

- La rappresentazione del suo modulo $|x|$ se $x \geq 0$
- La rappresentazione di $b^n - |x|$ se $x < 0$

- Esempio con 4 cifre binarie abbiamo

binario	decimale segno	binario	decimale segno	binario	decimale segno	binario	decimale segno
0000	0	0100	+4	1000	-8	1100	-4
0001	+1	0101	+5	1001	-7	1101	-3
0010	+2	0110	+6	1010	-6	1110	-2
0011	+3	0111	+7	1011	-5	1111	-1

Interi: complemento alla base

Bizzarro ma...

- Per i positivi la cifra più significativa è 0, per i negativi 1
- Gli algoritmi sono più semplici
  - 6 - 7 = 0110 + 1001 = 1111 = -1
  - -1 -5 = 1111 + 1011 = 11010 = -6

binario	decimale segno	binario	decimale segno	binario	decimale segno	binario	decimale segno
0000	0	0100	+4	1000	-8	1100	-4
0001	+1	0101	+5	1001	-7	1101	-3
0010	+2	0110	+6	1010	-6	1110	-2
0011	+3	0111	+7	1011	-5	1111	-1

Interi: complemento alla base

Bizzarro ma...

- Esempio: l'addizione si ottiene sommando le rappresentazioni e ragionando in modulo
  - 6 - 7 = 0110 + 1001 = 1111 = -1
  - -1 -5 = 1111 + 1011 = 11010 = -6

- Rende molto più semplice la realizzazione ad hardware (ALU)
- È la rappresentazione usata dai vari tipi **`int`** del C e di molti altri linguaggi

Complementazione

In generale come si trova la rappresentazione di $x < 0$ data quella di $|x|$?

- Ad esempio la rappresentazione in complemento a 2 di -2 data quella di 2 (0010) ?
- Per definizione, se utilizzo la base b e n cifre allora

$$-x = b^n - |x|$$

- Quindi basta trovare un algoritmo semplice per trivare le cifre del risultato della sottrazione a partire da quelle di $|x|$
- Vediamo quelle per la rappresentazione in base 2 senza entrare nel dettaglio della dimostrazione formale

Complementazione

Se ho *n* cifre e rappresentazione in base 2..

$$-x = 2^n - |x|$$

- Ad esempio con 4 cifre: rappresentiamo -6 e -2

2^4=16	10000
6	0110
10 (-6)	1010

2^4=16	10000
2	0010
14 (-2)	1110

Quindi basta:

- Ricopiare le cifre meno significative fino al primo 1 (compreso)
- Invertire tutte le altre più significative

Interi: complemento alla base

In generale

- Con n cifre in base b
- Si possono rappresentare gli interi nell'intervallo

$$\left[-\frac{b^n}{2}, \frac{b^n}{2} - 1\right]$$

- Se consideriamo la base 2 abbiamo come intervallo di rappresentazione:

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

- Valgono quindi le stesse considerazioni fatte per gli interi riguardo a overflow e ordine di applicazione degli operatori .....

Come si trova l'overflow ?

- Vediamo meglio come avviene l'addizione con rappresentazione in complemento a 2 con n cifre
  - Vengono sommati bit a bit i due numeri a partire dalla cifra meno significativa (posizione 0)
  - Sia r_{n-1} il riporto dalla posizione più significativa (la $n-1$) e r_{n-2} il penultimo riporto (posizione $n-2$)
  - È possibile dimostrare che si ha overflow solo se i due riporti sono diversi cioè $r_{n-1} \neq r_{n-2}$

- Vediamo alcuni esempi

Overflow: esempi

Non c'è overflow

rip	**0**	**0**	0	1	1	
+9		0	1	0	0	1
+3		0	0	0	1	1
+12		0	1	1	0	0

rip	**1**	**1**	1	1	1	
-9		1	0	1	1	1
-3		1	1	1	0	1
-12		1	0	1	0	0

C'è overflow

rip	**0**	**1**	0	0	0	
+9		0	1	0	0	1
+8		0	1	0	0	0
+17		1	0	0	0	1

rip	**1**	**0**	0	0	0	
-9		1	0	1	1	1
-8		1	1	0	0	0
-17		0	1	1	1	1

5 cifre intervallo di rappresentazione da -16 a +15

Numeri frazionari

- I numeri frazionari fra 0 ed 1 si rappresentano comunemente in base *b* come:

$$N_b = 0.c_{-1}c_{-2}\cdots c_{-n}$$

  - Al solito il peso delle cifre è determinato dalla loro posizione e dalla base *b* utilizzata

$$b^{-1} * c_{-1} + \cdots + b^{-n} * c_{-n} = \sum_{i=-n}^{-1} b^i * c_i = N_b$$

  - Ad esempio, 0.587 in base 10, corrisponde a

$$10^{-1} * 5 + 10^{-2} * 8 + 10^{-3} * 7$$

Numeri frazionari

- Usiamo la definizione per convertire in decimale un numero frazionario binario:

$$N_b = \sum_{i=-n}^{-1} b^i * c_i$$

- Ad esempio, 0.1011 in base 2, corrisponde a $2^{-1} * 1 + 2^{-3} * 1 + 2^{-4} * 1 = 0.6875$

Numeri frazionari

- Anche nel caso dei numeri frazionari dato il numero di cifre *n*, il numero di configurazioni rappresentabili è finito:
  - Si introducono degli errori di rappresentazione
  - L'errore è sempre maggiore di b^{-n} dove b è la base di rappresentazione

- Si hanno di nuovo problemi di non rappresentabilità del risultato di una operazione

Conversione da base 10 a base 2

- Il metodo piu semplice consiste nell'effettuare una sequenza di moltiplicazioni per 2 prendendo ad ogni passo la parte intera del risultato come cifra binaria della rappresentazione
- Esempio 1: convertiamo 0.125 da base 10 a base 2

0.125 * 2	0.25	**0**
0.25 * 2	0.5	**0**
0.5 * 2	1.0	**1**

- Il risultato è 0.001 e la rappresentazione è esatta con 3 cifre decimali

Conversione da base 10 a base 2

- Esempio 2: convertiamo 0.587 da base 10 a base 2

0.587 * 2	1.174	**1**
0.174 * 2	0.348	**0**
0.348* 2	0.696	**0**
0.696*2	1.392	**1**
0.393*2	0.784	**0**
0.784*2	1.568	**1**
0.568*2	....	**....**
.....	.....	**....**

- In questo caso la rappresentazione fino a 6 cifre è approssimata, l'accuratezza dipende dalle cifre disponibili: 0.1001 (4 cifre), 0.100101 (6 cifre)

Rappresentazione dei reali

- L'insieme e' infinito e dobbiamo fare delle approssimazioni, vedremo due metodi:
  - Virgola fissa
  - Virgola mobile

Reali: virgola fissa

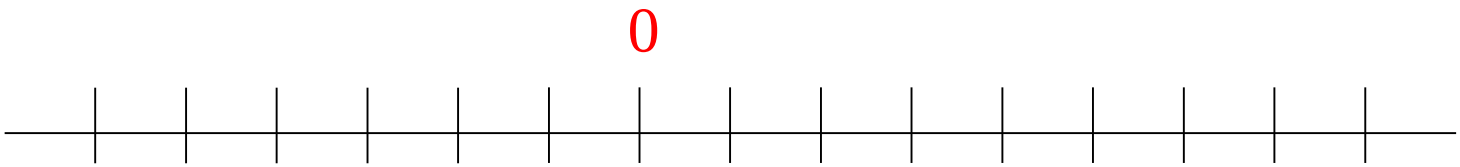
- Dividiamo le cifre disponibili fra la parte intera e la parte frazionaria, es.

$$N_b = c_{n-1}c_{n-2}\cdots c_0.\, c_{-1}c_{-2}\cdots c_{-m}$$

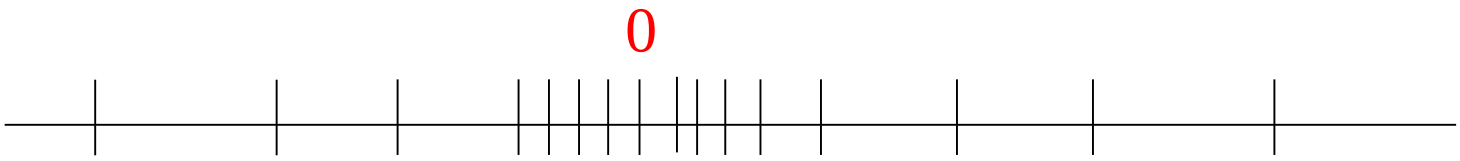
$$b^{n-1} * c_{n-1} + \cdots + b^0 * c_0 + b^{-1} * c_{-1} + \cdots + b^{-n} * c_{-n}$$

- m ed n sono gli stessi per tutti i numeri rappresentati

Reali virgola mobile



- Con la virgola fissa la precisione è sempre la stessa per tutti i reali sia in prossimità dello zero che altrove
- Sarebbe più ragionevole usare più precisione per valori più piccoli e meno per valori maggiori, graficamente ...



- La virgola mobile realizza questo con una rappresentazione esponenziale

Reali: virgola mobile

Nella rappresentazione esponenizale in base b

$$N_b = m * b^e$$

Ad esempio:

- $1150 = 0.115 * 10^4$
- Ma anche $1150 = 11.5 * 10^2$
- .....

Reali: virgola mobile

- Usa la rappresentazione in forma normalizzata in base b

$$N_b = m * b^e$$

Dove

- m è la mantissa che deve stare nell'intervallo $\frac{1}{b} \leq m < 1$
- ed e è la caratteristica, un intero con segno

Ad esempio la rappresentazione normalizzata di 1150 in base 10 è :

- $1150 = 0.115 * 10^4$

Reali: virgola mobile

Nella forma normalizzata in base b la mantissa ha la forma:

$$m = 0.c_{-1} \cdots c_{-k}$$

- Quindi nel caso della base 2 servono k bit
- E il valore della mantissa è limitato da

$$(0.1)\frac{1}{b} \leq m \leq \sum_{i=-k}^{-1} b^i * c_i \; (0.111 \ldots 1)$$

Reali: virgola mobile

- Anche la caratteristica è limitata

$$N_b = m * b^e$$

  - Se assumiamo h bit
  - E la rappresentazione in complemento a 2
  - L'intervallo di rappresentazione della caratteristica è:

$$-2^h \leq e \leq 2^h - 1$$

- Questi due vincoli fissano il minimo e massimo numero rappresentabile
  - Tuttavia in questo caso anche internamente a questo intevallo la rappresentazione è approssimata

Reali: virgola mobile

- Analizziamo meglio i problemi della rappresentazione
  - Vengono rappresentati solo numeri razionali
  - I reali sono approssimati con il razionale più vicino
  - I numeri rappresentati sono più densi vicino allo 0 e più distanti vicino al massimo e minimo rappresentabile
    - Quindi l'errore che commettiamo non è fisso ma dipende dal numero che stiamo rappresentando
  - L'insieme non è chiuso rispoetto a +,* ...

Reali in virgola mobile

Esistono quindi diversi errori

- Errore di *underflow* (siamo arrivati troppo vicino a 0, essenzialmente la mantissa si azzera)
- Errore di *overflow* (numero troppo elevato, la caratteristica va in overflow)
- Errore di *arrotondamento* (vengono tralasciate delle cifre significative del numero rappresentato)

Reali: virgola mobile

- Esempio tipico su 32 bit
  - 1 bit per il segno (0 positivo, 1 negativo)
  - 7 per la caratteristica (in complemento)
  - 24 per la mantissa
- IEEE ha standard su 32,64,.... Bit
  - Si usa la rappresntazione in eccesso (es. eccesso 127 su 8 bit) invece che quella in complemento a 2 per *e*
  - Convenzioni speciali per rappresentare infinito negativo e positivo
  - Convenzioni sul tipo di arrotondamento/troncamento da effettuare nelle operazioni aritmetiche e nelle conversioni