Architettura degli Elaboratori
3 - Rappresentazione dell'Informazione

Ugo Dal Lago

Dipartimento di Scienze dell'Informazione
Università degli Studi di Bologna

Anno Accademico 2007/2008

Sommario

Rappresentazione delle Informazioni

- I calcolatori gestiscono **informazioni** di varia natura: testi, immagini, suoni, filmati, etc.
- I calcolatori, però, lavorano unicamente con sequenze di bit.
- Risulta quindi necessaria di un'opportuna codifica.
- Esamineremo le codifiche dei dati più semplici: **numeri** e **caratteri**.
- Gli schemi di codifica devono essere:
  - **compatti**, ovvero non devono utilizzare rappresentazioni troppo più lunghe dello stretto necessario.
  - **pratici**, ovvero devono facilitare la computazione.
  - **fedeli**, ovvero devono evitare di perdere informazione.

Codifica: Teoria Generale

- Partiamo da un insieme di **dati rappresentabili** D.
- Abbiamo a disposizione un **alfabeto** finito A.
- Consideriamo **parole** nell'alfabeto A, ovvero sequenze di simboli nell'alfabeto. Indichiamo con A^* l'insieme di tutte le parole finite nell'alfabeto A.
- Una **funzione di codifica** $D \rightarrow A^*$ è una mappa iniettiva dallo spazio dei dati nell'insieme delle parole dell'alfabeto.
- Se A contiene k simboli, con parole di lunghezza n si possono rappresentare k^n oggetti diversi.
- Esempi:
  - D_N è l'insieme dei numeri naturali, $A_{10} = \{0, 1, 2, \ldots, 9\}$ è l'alfabeto e la funzione di codifica associa ad ogni numero naturale la sua rappresentazione in base 10.
  - D_N è l'insieme dei numeri naturali, $A_2 = \{0, 1\}$ è l'alfabeto e la funzione di codifica associa ad ogni numero naturale la sua rappresentazione in base 2.
  - D_P è l'insieme delle parole in uso nella lingua italiana, $A_P = \{a, b, c, \ldots, u, v, z, à, è, é, ì, ò, ù\}$ e la funzione di codifica associa ad ogni parola alla sequenza di lettere corrispondente.

Aritmetica Finita

- Nei sistemi di calcolo, l'alfabeto di riferimento è $A_2 = \{0, 1\}$.
- Inoltre, fissato un sistema di calcolo, la quantità di memoria a disposizione sarà sempre **finita**, e sarà quindi possibile avere a disposizione solo le sequenze in A_2^* lunghe al più k.
- Ciò significa, in particolare, che in un tipico sistema di calcolo non è possibile tener traccia di tutti i numeri naturali!
- Di conseguenza, occorre considerare sottoinsiemi finiti dell'insieme D_N dei numeri naturali: D_N^k sarà l'insieme dei numeri naturali rappresentabili con k cifre.
  - D_N^3 contiene 156, 932,...
  - D_N^8 contiene tutti i numeri naturali inferiori al miliardo.
- D_N^k non è mai chiuso rispetto alle operazioni aritmetiche $+, -$ oltre che a $\times, /$.
  - Possiamo ottenere numeri troppo grandi (**overflow**), oppure troppo piccoli (**underflow**).

Sistemi di Numerazione in Base Fissa

- Da qui in poi, studieremo i sistemi di numerazione posizionale.
- La codifica più comune per i numeri interi è la codifica **decimale** posizionale, in cui l'alfabeto utilizzato è $A_{10} = \{0, 1, 2, \ldots, 9\}$.
- La generica sequenza $d_k \ldots d_2 d_1 d_0$ codifica il numero intero:

$$\sum_{i=0}^{k} d_i \times 10^i$$

- In informatica, altre codifiche posizionali sono più importanti:
  - La codifica **binaria**, in cui l'alfabeto utilizzato è $A_2 = \{0, 1\}$.
  - La codifica **ottale**, in cui l'alfabeto utilizzato è $A_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$
  - La codifica **esadecimale**, in cui l'alfabeto utilizzato è $A_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. I simboli A, B, C, D, E, F corrispondono a $10, 11, 12, 13, 14, 15$, rispettivamente.

Il Numero 2001 in Varie Basi

Binary	1	1	1	1	1	0	1	0	0	0	1
	1×2^{10}	$+ 1 \times 2^9$	$+ 1 \times 2^8$	$+ 1 \times 2^7$	$+ 1 \times 2^6$	$+ 0 \times 2^5$	$+ 1 \times 2^4$	$+ 0 \times 2^3$	$+ 0 \times 2^2$	$+ 0 \times 2^1$	$+ 1 \times 2^0$
	1024	+ 512	+ 256	+ 128	+ 64	+ 0	+16	+ 0	+ 0	+ 0	+ 1
Octal	3	7	2	1							
	3×8^3	$+ 7 \times 8^2$	$+ 2 \times 8^1$	$+ 1 \times 8^0$							
	1536	+ 448	+ 16	+ 1							
Decimal	2	0	0	1							
	2×10^3	$+ 0 \times 10^2$	$+ 0 \times 10^1$	$+ 1 \times 10^0$							
	2000	+ 0	+ 0	+ 1							
Hexadecimal	7	D	1								
	7×16^2	$+ 13 \times 16^1$	$+ 1 \times 16^0$								
	1792	+ 208	+ 1								

Conversione tra Basi

- Due parole (in alfabeti diversi) sono **equivalenti** se codificano lo stesso numero naturale.
- **Convertire** una sequenza significa trovare una parola (in un alfabeto diverso) equivalente alla prima.
- Convertire un numero in notazione ottale o esadecimale in notazione binaria (o viceversa) è molto semplice.
- Se si parte da un numero in notazione decimale e lo si vuole convertire in notazione binaria si può procedere per **divisioni successive**.
- Viceversa, un numero in notazione binaria può essere convertito in notazione decimale procedendo per **moltiplicazioni successive**.
- Comunque, è sempre possibile utilizzare la definizione vista in precedenza: una sequenza $s_k \ldots s_2 s_1 s_0$ in base b codifica il numero naturale

$$\sum_{i=0}^{k} s_i \times b^i$$

Esempi di Conversione

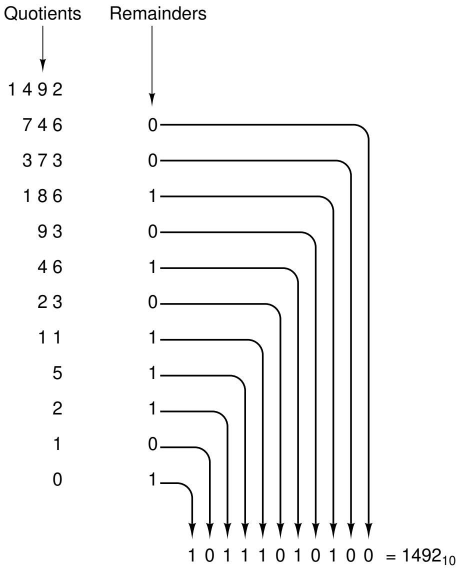
Example 1

Hexadecimal	1 9 4 8 . B 6
Binary	0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 . 1 0 1 1 0 1 1 0 0
Octal	1 4 5 1 0 . 5 5 4

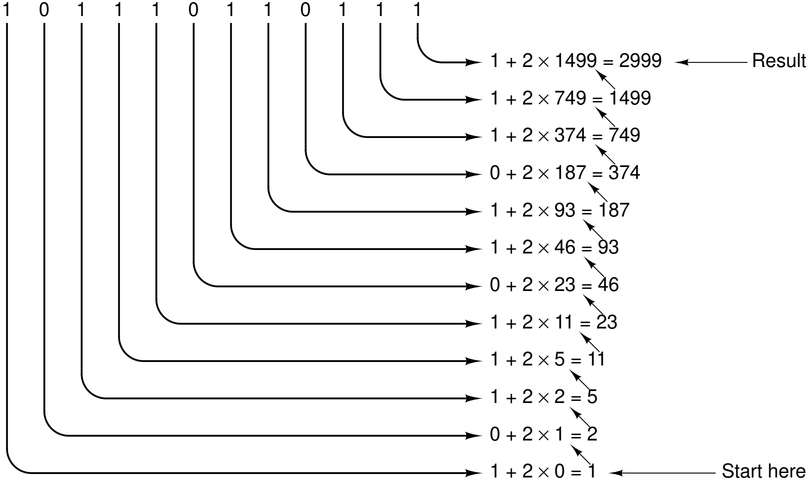
Example 2

Hexadecimal	7 B A 3 . B C 4
Binary	0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 . 1 0 1 1 1 1 0 0 0 1 0 0
Octal	7 5 6 4 3 . 5 7 0 4

Esempi di Conversione



Esempi di Conversione



Numeri Binari Negativi

- Esistono un certo numero di schemi di codifica per la rappresentazione dei numeri binari con segno.
- Nello schema detto **modulo e segno**, si utilizza il bit più significativo (quello più a sinistra) come bit di segno (0 per il $+$ e 1 per il $-$).
- Anche nel sistema detto **complemento a uno** si usa il bit di segno. Nei numeri negativi, però, occorre invertire tutti gli altri bit.
- Nel sistema detto **complemento a due** si usa sempre il bit di segno. Per i numeri negativi, occorre invertire gli altri bit, e poi si aggiunge 1 al risultato.
- Esiste poi il sistema **in eccesso di** 2^{k-1} per numeri di k bit. Ogni numero (positivo o negativo) viene rappresentato dal numero binario ottenuto aggiungendo 2^{k-1} al numero di partenza.

Numeri Binari Negativi

- Ad esempio, se si vuole rappresentare -6 con 8 bit in complemento a due, si parte da 6 in binario:

$$00000110,$$

si passa a -6 in complemento a uno:

$$11111001,$$

e si arriva a -6 in complemento a due:

$$11111010.$$

- Data una stringa in complemento a due, diciamo $s_k s_{k-1} \ldots s_1 s_0$, il numero da essa rappresentato è ottenuto mediante la formula:

$$-s_k \times 2^k + \sum_{i=0}^{k-1} s_i \times 2^i$$

Numeri Binari

- L'intervallo di numeri rappresentabili dipende dal sistema utilizzato e dal numero di bit a disposizione. Supponiamo nel seguito di utilizzare k bit.
- La dimensione dell'intervallo rappresentabile cresce esponenzialmente con k.
- Se ci interessano solo i numeri naturali, possiamo rappresentare l'intervallo compreso tra 0 e $2^k - 1$.
  - Se $k = 8$ l'intervallo sarà quello tra 0 e 255.
- Se si utilizza la codifica **modulo e segno** o la codifica **complemento a uno** si potranno rappresentare i numeri tra $-2^{k-1} + 1$ e $2^{k-1} - 1$. In particolare, ci saranno due stringhe binarie entrambe corrispondenti a 0.
  - Se $k = 8$ l'intervallo sarà quello tra -127 e $+127$.
- Se si utilizza lacodifica **complemento a due** o la codifica ad **eccesso** si potranno rappresentare i numeri tra -2^{k-1} e $2^{k-1} - 1$, visto che una sola stringa binaria corrisponde a 0.
  - Se $k = 8$ l'intervallo sarà quello tra -128 e $+127$.

Somma Binaria

Addendo	$0+$	$0+$	$1+$	$1+$
Addendo	$0=$	$1=$	$0=$	$1=$
Somma	0	1	1	0
Riporto	0	0	0	1

Aritmetica Binaria

- La somma aritmetica tra due numeri in notazione binaria si effettua seguendo un procedimento in uso nell'aritmetica decimale.
- Nell'aritmetica in **complemento a uno**, il riporto generato dai due bit più significativi viene sommato al bit meno significativo.
- Nell'aritmetica in **complemento a due**, il riporto generato dalla somma dei bit più significativi viene semplicemente scartato.
- Per quanto riguarda l'**overflow**:
  - Se gli addendi hanno segno opposto non si è verificato overflow.
  - Se gli addendi hanno lo stesso segno e il risultato ha segno opposto, si è verificato overflow.
  - Se gli addendi e il risultato hanno tutti lo stesso segno, si ha overflow se e solo se il risultato in corrispondenza del bit di segno differisce dal riporto generato oltre il bit di segno.

Somma in Complemento a Uno e in Complemento a Due

Decimal	1's complement	2's complement
10	00001010	00001010
+ (–3)	11111100	11111101
+7	1 00000110	1 00000111
	carry 1	discarded
	00000111	

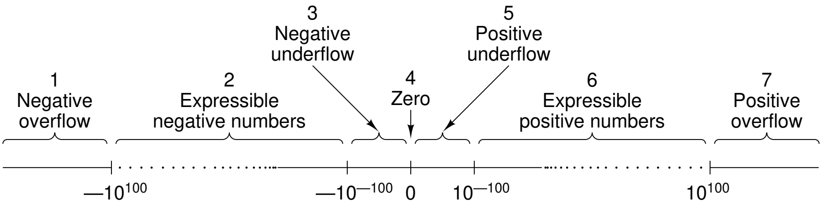
Rappresentare Numeri con Virgola

- Generalizzare i sistemi che abbiamo introdotto a **numeri con virgola** non è difficile.
  - La posizione della virgola è fissata a priori, altrimenti risulta impossibile eseguire operazioni aritmetiche in precisione finita.
- Spesso l'**intervallo** dei numeri utilizzati è molto grande.
  - In tal caso, l'impiego di numeri con virgola fissa diventa problematico. Avremmo bisogno di tantissimi bit, anche se le cifre significative non sono così tante.
- Abbiamo bisogno di sistema di rappresentazione dei numeri nel quale l'intervallo dei numeri rappresentabili non dipenda dal numero di cifre significative
  - Abbiamo bisogno dei numeri con **virgola mobile**!

Numeri in Virgola Mobile

- Un modo per rappresentare il numer n disaccoppiando l'intervallo dalla precisione consiste nell'esprimere il numero n come segue:
$$n = f \times 10^e$$
dove f è chiamato **frazione** o **mantissa** e e è chiamato **esponente** o **caratteristica**.
- Esempi:
$$\begin{array}{lclcl} 3,14 & = & 0,314 \times 10^1 & = & 3,14 \times 10^0 \\ 0,000001 & = & 0,1 \times 10^{-5} & = & 1,0 \times 10^{-6} \\ 1941 & = & 0,1941 \times 10^4 & = & 1,941 \times 10^3 \end{array}$$
- Supponiamo di rappresentare:
  - La frazione f con 0 oppure con un valore con segno a tre cifre tale che $0,1 \leq |f| < 1$.
  - L'esponente è un valore con segno a due cifre.
- Queste rappresentazioni variano, in modulo, da $0,1 \times 10^{-99}$ a $0,999 \times 10^{99}$, ma richiedono solamente cinque cifre decimali!

La Retta Reale e i Numeri in Virgola Mobile



Numeri in Virgola Mobile

- Un'operazione che coinvolga due numeri in virgola mobile può produrre:
  - Un **errore di overflow**, se il risultato è troppo grande per essere espresso in virgola mobile.
  - Un **errore di underflow**, se il risultato è troppo piccolo per essere espresso in virgola mobile.
- Sappiamo poi che:
  - I numeri reali sono **densi**: dati due reali x e y tali che $x < y$ esiste un altro numero reale z con $x < z < y$.
  - I numeri in virgola mobile sono un sistema **discreto**, che può esprimere una quantità finita di numeri reali. Per esempio dividendo $0,100 \times 10^3$ per 3 otteniamo un reale che non è esprimibile nel sistema e quindi va **arrotondato**.
- Le considerazioni appena fatte valgono anche quando il numero di cifre riservate alla mantissa e il numero di cifre riservate all'esponente cambiano.
- In concreto, mantissa ed esponente sono rappresentati da cifre binarie. In tal caso parliamo di mantissa **normalizzata** quando la cifra più significativa è 1.

Numeri in Virgola Mobile: Esempi

Example 1: Exponentiation to the base 2

2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} 2^{-6} 2^{-7} 2^{-8} 2^{-9} 2^{-10} 2^{-11} 2^{-12} 2^{-13} 2^{-14} 2^{-15} 2^{-16}

Unnormalized: $\underbrace{\mathbf{0}}_{\substack{\text{Sign}\\ +}}\ \underbrace{\mathbf{1010100}}_{\substack{\text{Excess 64}\\ \text{exponent is}\\ 84-64=20}}\,.\,\underbrace{\mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1}}_{\substack{\text{Fraction is } 1\times 2^{-12} + 1\times 2^{-13}\\ +1\times 2^{-15} + 1\times 2^{-16}}}$ $= 2^{20}\,(1\times 2^{-12} + 1\times 2^{-13} + 1\times 2^{-15} + 1\times 2^{-16}) = 432$

To normalize, shift the fraction left 11 bits and subtract 11 from the exponent.

Normalized: $\underbrace{\mathbf{0}}_{\substack{\text{Sign}\\ +}}\ \underbrace{\mathbf{1001001}}_{\substack{\text{Excess 64}\\ \text{exponent is}\\ 73-64=9}}\,.\,\underbrace{\mathbf{1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}}_{\substack{\text{Fraction is } 1\times 2^{-1} + 1\times 2^{-2}\\ +1\times 2^{-4} + 1\times 2^{-5}}}$ $= 2^{9}\,(1\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-4} + 1\times 2^{-5}) = 432$

Example 2: Exponentiation to the base 16

16^{-1} 16^{-2} 16^{-3} 16^{-4}

Unnormalized: $\underbrace{\mathbf{0}}_{\substack{\text{Sign}\\ +}}\ \underbrace{\mathbf{1000101}}_{\substack{\text{Excess 64}\\ \text{exponent is}\\ 69-64=5}}\,.\,\underbrace{\mathbf{0000\quad 0000\quad 0001\quad 1011}}_{\text{Fraction is } 1\times 16^{-3} + B\times 16^{-4}}$ $= 16^{5}\,(1\times 16^{-3} + B\times 16^{-4}) = 432$

To normalize, shift the fraction left 2 hexadecimal digits, and subtract 2 from the exponent.

Normalized: $\underbrace{\mathbf{0}}_{\substack{\text{Sign}\\ +}}\ \underbrace{\mathbf{1000011}}_{\substack{\text{Excess 64}\\ \text{exponent is}\\ 67-64=3}}\,.\,\underbrace{\mathbf{0001\quad 1011\quad 0000\quad 0000}}_{\text{Fraction is } 1\times 16^{-1} + B\times 16^{-2}}$ $= 16^{3}\,(1\times 16^{-1} + B\times 16^{-2}) = 432$

Codifica dei Caratteri

- Quando l'insieme dei dati da rappresentare è l'insieme delle parole in un certo **linguaggio naturale**, occorre prima di tutto capire come si codificano **i singoli caratteri**, le singole lettere.
- Il primo tentativo fu il codice **ASCII** (American Standard Code for Information Interchange).
  - Ogni carattere è codificato da un intero a 7 bit.
  - In totale, quindi, abbiamo 128 caratteri distinti.
- Diversi linguaggi naturali, però, utilizzano insiemi di caratteri diversi. Per questo motivo si introdusse il concetto di **code page**,
  - Un insieme di 256 caratteri specifico di una lingua o di un gruppo di lingue.
- Il codice destinato a diventare lo standard è chiamato **UNICODE**.
  - Ad ogni carattere è associata una sequenza di 16 bit chiamata **code point**.
  - A ciascun alfabeto è associato un insieme di code point tra loro consecutivi (nel senso degli interi corrispondenti ai code point).