Rappresentazione dell'informazione

I calcolatori gestiscono dati di varia natura: testi, immagini, suoni, filmati,

nei calcolatori rappresentati con sequenze di bit:

mediante un'opportuna codifica

presentiamo le codifiche dei dati gestite dall'hardware: numeri (e caratteri).

Argomenti trattati:

- numeri: naturali, interi, reali e le operazioni aritmetiche nell'hardware,
- caratteri: diversi codici di rappresentazione,
- codici di correzione degli errori,
- organizzazione della memoria,

Codifica: Teoria generale

- insieme dei dati rappresentabili (D),
- alfabeto ($A = \{0, 1\}$): insieme di simboli,
- codifica ($D \to A^*$): mappa tra dati e le sequenze di bit

In una codifica di lunghezza costante, con n bit si rappresentano sino a 2^n dati diversi.

Esistono codifiche a lunghezza variabile (dati diversi occupano un diverso numero di bit).

Proprietà di una codifica:

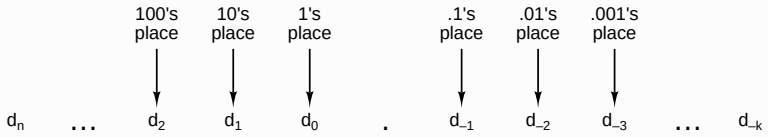
- compatta: si limita in numero di byte necessari,
- pratica: facilitare la computazione,
- fedele: codifiche senza perdita di informazione, o con poca informazione persa.

Esigenze contrapposte: si cerca un compromesso.

L'aritmetica dei calcolatori

- come codificare i numeri: naturali, interi, reali.
- come eseguire le operazioni

Notazione posizionale: il peso di una cifra dipende dalla sua posizione:



$$\text{Number} = \sum_{i=-k}^{n} d_i \times 10^i$$

La stessa idea con *basi* diverse

Binary 1 1 1 1 1 0 1 0 0 0

$1\times2^{10} + 1\times2^9 + 1\times2^8 + 1\times2^7 + 1\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1$

1024 + 512 + 256 + 128 + 64 + 0 +16 + 0 + 0 + 0

Octal 3 7 2 1

$3\times8^3 + 7\times8^2 + 2\times8^1 + 1\times8^0$

1536 + 448 + 16 + 1

Decimal 2 0 0 1

$2\times10^3 + 0\times10^2 + 0\times10^1 + 1\times10^0$

2000 + 0 + 0 + 1

Hexadecimal 7 D 1 .

$7\times16^2 + 13\times16^1 + 1\times16^0$

1792 + 208 + 1

Notazione binaria

Il calcolatore utilizza base 2

- un segnale digitale rappresenta una cifra
- semplificazione dell'hardware

Alternativa: BCD (Binary Coded Decimal)

- si rappresenta una sequenza di cifre decimali,
- ogni cifra rappresentata da 4 bit,
- vantaggi nessun arrotondamento per numeri con virgola,
- svantaggi complica i circuiti per le operazioni, usa più bit dello stretto necessario.

Conversioni di base: da 2 a 10

Due possibili metodi (algoritmi).

1) Sommare il peso delle singole cifre.

$101110110111_{two} =$

$2^{11} + 2^9 + 2^8 + 2^7 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0$

$= 2048+512+256+128+32+16+4+2+1 = 2999_{ten}$

Intuitivo ma poco efficiente.

Ambiguità

"Esistono 10 tipi di persone: quelle che capiscono la notazione binaria e quelli che non la capiscono."

Se si usano diverse basi, bisogna specificare la base utilizzata.
Diverse modi, ma in genere:
un pedice alla fine della sequenza di cifre.

Esempi: 257_{ten}, $(257)_{10}$, 257_{eight}, $257H$.

Alternative: lettere all'inizio della sequenza di cifre:
$H328C$, $HABCD$, $0xABCD$, $O127$

Secondo metodo di conversione

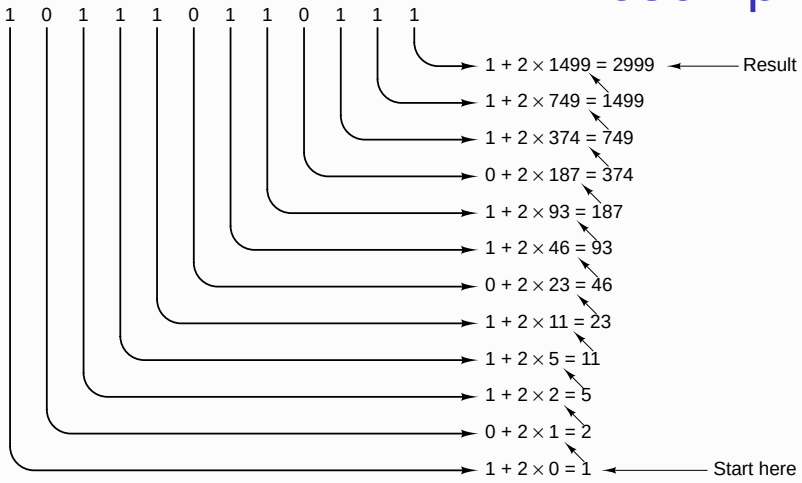
Si convertono le sottosequenze iniziali della sequenza.

Per convertire 101110110111_{two}:

$1_{two} = 1_{ten}$
$10_{two} = 2_{ten}$
$101_{two} = 5_{ten} = 2 * 2 + 1$
$1011_{two} = 11_{ten} = 5 * 2 + 1$

Ad ogni passo:
"nuovo valore" =
"valore precedente" $*$ 2 + "nuova cifra"

2^o metodo di conversione: esempio



Metodo applicabile per convertire da una qualsiasi base in base 10.

Conversione da base 10 a base 2

Due possibili metodi (algoritmi):

1) Scomporre in potenze di 2

$1492 = 1024 + 468 = 1024 + (256 + 212)$
$= 1024 + (256 + (128 + 84) \ldots$
$= 1024 + 256 + 128 + 64 + 16 + 4$
$= 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^2$

Metodo intuitivo ma poco pratico.

Altro metodo: serie di divisioni - resti

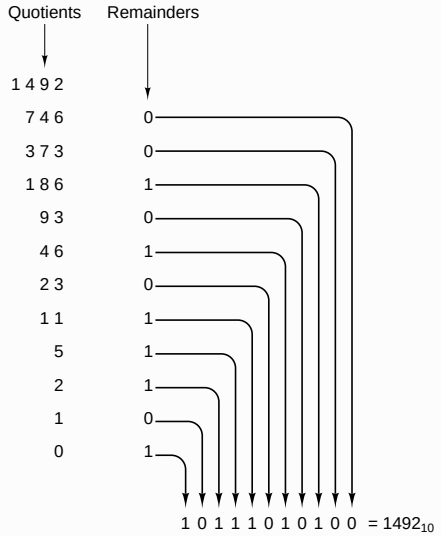
Dividendo un numero (decimale) per 2 ottengo:

- l'ultima cifra binaria,
- il numero rappresentato dalle rimanenti cifre.

Con un serie di divisioni per 2 valuto tutte le cifre.

Metodo applicabile per convertire da base dieci ad una qualsiasi base (serie di divisione per la "base").

Serie di divisioni - resti: esempio



Base 8 (ottale) o 16 (esadecimale)

A volte necessario rappresentare numeri binari di decine di cifre.
Rappresentazioni ottale e esadecimale sono comode:

- un insieme di cifre simile a quello della notazione decimale;
- più compatte della notazione binaria;
- esiste un metodo semplice e veloce per convertire di base.

La notazione esadecimale richiede cifre extra:
A (10), B (11), C (12), D (13), E (14), F (15).

Conversioni

Da base 2 a base 16 (8)

- dividere la sequenza di cifre in gruppi di 4 (3),
- trasformare ciascun gruppo in una cifra.

Da base 16 (8) a base 2

- trasformare ciascuna cifra in sequenza di 4 (3) cifre binarie,
- riunire le sequenze così ottenute.

Esempi

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

Una argomento per la correttezza

$$
\begin{aligned}
&(d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_{\text{two}} = \\
&= 2^7 d_7 + 2^6 d_6 + 2^5 d_5 + 2^4 d_4 + 2^3 d_3 + 2^2 d_2 + 2^1 d_1 + 2^0 d_0 \\
&= (2d_7 + d_6)2^6 + (4d_5 + 2d_4 + d_3)2^3 + (4d_2 + d_1 + d_0) \\
&= (2d_7 + d_6)2^{3*2} + (4d_5 + 2d_4 + d_3)2^{3*1} + (4d_2 + d_1 + d_0) \\
&= (2d_7 + d_6)8^2 + (4d_5 + 2d_4 + d_3)8^1 + (4d_2 + 2d_1 + d_0) \\
&= ((2d_7 + d_6)(4d_5 + 2d_4 + d_3)(4d_2 + 2d_1 + d_0))_{\text{eight}}
\end{aligned}
$$

L'aritmetica del calcolatore

Tre diverse classi di numeri: naturali (unsigned), interi (signed), reali (floating point).

Rappresentati con un numero fisso di cifre.

- Naturali e interi: 16 – 32 – 64 bit.
- Reali (frazionari): 32 – 64 – 128.

Conseguenze.

- L'insieme dei numeri rappresentabili è limitato. Massimo rappresentabile.
- Per i numeri reali, il calcolatore da risultati approssimati. (le rappresentazioni sono approssimate, le operazioni introducono errori).
- Più formati per numeri e operazioni.

Errore di overflow

Errore di overflow: il risultato di un operazione è più grande del massimo valore rappresentabile.

Esempio: la somma o il prodotto di due numeri di cui uno uguale al massimo numero rappresentabile.

L'hardware segnala eventuali gli errori di overflow,

Esempio: nella somma di due numeri naturali, l'hardware controlla se le cifre più significative generano un riporto.

Operazioni aritmetiche nell'hardware

L'hardware implementa le 4 operazioni aritmetiche.
Gli algoritmi (metodi) validi per base dieci restano validi per base due.
Per motivi di efficienza, l'hardware usa algoritmi più sofisticati.

Addizione:

- somme e riporti;
- in hardware: bisogna gestire il problema della propagazione del riporto.

Operazioni aritmetiche

Sottrazione:

- sottrazioni e prestiti;
- in hardware: si calcola l'opposto e esegue un addizione,
  soluzione utile perché viene usata la notazione *complemento a due* per i negativi.

Moltiplicazione:

- serie di somme;
- in hardware: stessa idea ma parallelizzata, più somme contemporaneamente, meno cicli di somme (Dadda and Wallace multipliers).

Operazioni aritmetiche

Divisione:

- serie di sottrazioni;
- in hardware: serie di sottrazione o, per i numeri floating-point, calcolo dell'inverso e prodotto.

Interi: 4 rappresentazioni.

In hardware: 4 possibili alternative.

- segno e valore assoluto
  $9_{\text{dieci}} = 0\ 0001001$
  $-9_{\text{dieci}} = 1\ 0001001$
- complemento a 1
  $9_{\text{dieci}} = 0\ 0001001$
  $-9_{\text{dieci}} = 1\ 1110110$
- complemento a 2
  $9_{\text{dieci}} = 0\ 0001001$
  $-9_{\text{dieci}} = 1\ 1110111$
- eccesso (128)
  $9_{\text{dieci}} = 1\ 0001001 \quad (9 + 128 = 137)$
  $-9_{\text{dieci}} = 0\ 1110111 \quad (-9 + 128 = 119)$

Complemento a 2

La rappresentazione maggiormente usata.

- Motivazioni: semplifica l'ALU (Arithmetic Logic Unit).
- Definizione alternativa: con n cifre binarie, il numero negativo $-i$ viene rappresentato da $2^n - i$. I due valori sono congruenti modulo 2^n, $-i \equiv_{2^n} (-i + 2^n)$.
- Appunto: è una notazione per numeri sia negativi che positivi.

Operazioni in complemento a 2

Somma di interi:

- eseguo la somma come sei i numeri fossero numeri naturali scritti in notazione binaria.
- Perché funziona? Proprietà dell'aritmetica modulo.
  Il numero negativo $-i$ viene rappresentato da $2^n - i$
  $(-i \equiv_{2^n} 2^n - i)$.
  Se $j \equiv_{2^n} j'$, $k \equiv_{2^n} k'$
  allora $(j + k) \equiv_{2^n} (j' + k')$
- diverso il controllo dell'overflow: il segno degli addendi non coincide con quello del risultato.

Operazioni in complemento a 2

- Opposto: con un'operazione di complemento a 2 del numero (inverto le cifre e aggiungo uno).
- Estensione del numero di cifre (scrivo un numero con più cifre preservandone il valore): aggiungo cifre uguali alla cifra segno.
- Conversione in base dieci: Per convertire un numero negativo n: determino il valore assoluto calcolando l'opposto di n.

Numeri frazionari (con virgola)

I pesi delle cifre dopo la virgole sono potenze negative di due.

$$1,011 = 1*2^0+0*2^{-1}+1*2^{-2}+1*2^{-3} = 1+1/4+1/8$$

Solo i numeri nella forma $z/2^n$ sono rappresentabili con un numero finito di cifre;

esempio : $0,4$ scritto in base due diventa un numero periodico.

Metodi di conversione

Si convertono separatamente parte intera e parte frazionaria.

Conversione della parte frazionaria:

- da base due (o base b) a base dieci:
  calcolo la sommatoria delle cifre decimali moltiplicate per il loro peso.
- da base dieci a base due (o base b):
  cifre ottenute attraverso un ciclo,
  ad ogni iterazione:
  moltiplico parte frazionaria per 2 (b),
  - parte intera del risultato: nuova cifra del risultato;
  - parte frazionaria: valore da usare prossima iterazione.

Operazioni su numeri frazionari

Le tecniche usuali:

- Somma e sottrazione: sommo (sottraggo) cifre delle stesso peso.
- opero con i numeri senza virgola e poi valuto dove posizionare la virgola nel risultato.

Notazione floating-point

Per i numeri frazionari si usa la notazione floating-point (scientifica) (esponente– mantissa).
Il numero X è rappresentato da una coppia (m, e)

- $X = m \times 10^e$
- m: mantissa, numero frazionario, $(1 \leq |m| < 10)$,
- e: esponente, numero intero

Vantaggi: si rappresentano numeri grandi (e piccoli) sinteticamente, evitando gli zeri non significativi.

Floating-point nel calcolatore

Un numero reale X viene rappresentato da una sequenza di bit spezzata in due parti:

- una parte per la mantissa m,
  notazione normalizzata, la virgola in posizione fissa (es. dopo la prima cifra) non viene rappresentata,
- una parte per l'esponente e (numero intero)

$$X = m \times 2^e$$

L'esponente indica di quante posizioni deve essere spostata la virgola nella mantissa.

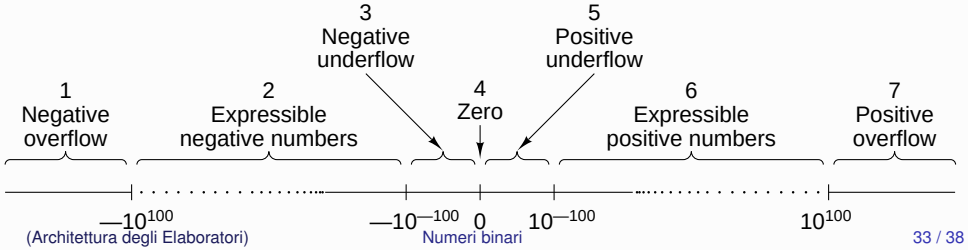
Floating-point nel calcolatore

Un numero fisso di bit per m ed e.

Un numero finito di valori rappresentabili.

A secondo di come partiziono le cifre:

- più bit per e $\Rightarrow$ più ampio l'intervallo
- più bit per m $\Rightarrow$ maggiore precisione (numeri più densi)



Computazione floating-point

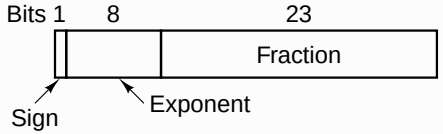
Difetti:

- imprecisioni nei calcoli: ogni operazione può generare un errore,
- overflow: risultati non rappresentabili, perché troppo grandi,
- underflow: risultati non rappresentabili, perché troppo vicini a 0, approssimati con 0.

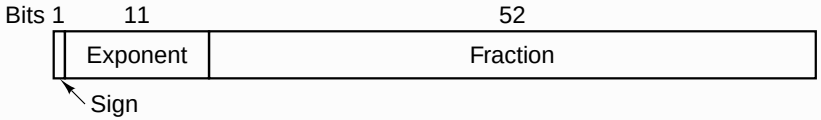
Notazione IEEE standard

- Fino agli anni 80: processori diversi usavano notazioni diverse, problemi nello scambio dati.
- Produttori informatici affidano a Williams Kahan la definizione di uno standard,
- notazione IEEE 754 standard.
- gestazione molto lunga,
- Usata in tutti i calcolatori,
- Turing award.

Notazione IEEE standard



(a)



(b)

segno 0: positivo, 1: negativo,
esponente eccesso 127 (1023)
mantissa notazione normalizzata, la mantissa inizia con 1, implicito (non viene rappresentato

Casi particolari

Nel caso l'esponente abbia valore minimo o massimo si usano altre regole.
Bit di esponente:

- 00000000: inizio implicito della mantissa: 0,; esponente: -126 (numeri denormalizzati).
- 11111111, con mantissa $00\ldots00$: infinito (gestisce l'overflow)
- 11111111, con mantissa diversa da $00\ldots00$: not a number (NaN), risultato non rappresentabile, (es. $0/0$)