La rappresentazione dei dati

Base binaria

- E’ la base minima che include cifre diverse
  - si devono conoscere le tabelline dello 0 dell’ I
    - in elettronica si realizzano bene dispositivi bistabili
- There are only 10 types of people in the world — those who understand binary, and those who don't.

Conversione della base di rappresentazione

- Due algoritmi di conversione:
  - si eseguono i conti nella base di arrivo
  - si eseguono i conti nella base di partenza

Conti nella base di arrivo

- Sviluppo in forma polinomiale nella base di partenza
- conversione di coefficienti e potenze nella base di arrivo
- somme e prodotti calcolati nella base di arrivo

  $[125]_{10} = [1*10*10+2*10+5]_{10} = [1*1010*1010+...]_2$

Conti nella base di partenza

- Si usa l'algoritmo dei resti successivi
  - pratico per convertire da base 10 a base 2
- La rappresentazione del numero A in base 2 è data da $[...0,0,a_N,a_{N-1},...a_1,a_0]_2$, determinata da :

$$A = \sum_{k=0}^{\infty} a_k 2^k = a_0 + 2\sum_{k=0}^{\infty} a_{k+1} 2^k$$

- a_0 = 1 se e solo se A è dispari, ovvero è il resto della divisione intera: $A/2 = \sum_{k=0}^{\infty} a_{k+1} 2^k$

- Continuando il procedimento per a_1 e successivi, si termina quando il quoziente della divisione è 0
  - E' conveniente mettere quozienti e resti su una diagonale (eseguendo i conti su carta)
- La sequenza dei resti sono i coefficienti a_k, in ordine inverso da LSB (a_0) a MSB (a_n)

Base esadecimale

- Codifica più compatta (0,...,9,A,...,F)
- Per convertire in base 2 si impacchettano i bit 4 a 4 (4bit = nibble):
  - $[125]_{10} = [0111\ 1101]_2 = [7D]_{16}$
- roses are #FF0000
  violets are #0000FF
  all my base
  are belong to you

Parti frazionarie

- Conto in base di arrivo:
  - 0.7 = 7 / 10 = 0111 / 1010
- Conto in base di partenza:
  - algoritmo di moltiplicazioni successive, duale delle divisioni successive: si moltiplica per due la parte frazionaria, sottraendone ogni volta la parte intera. La sequenza delle parti intere fornisce la rappresentazione.

Interi senza segno

- unsigned int
- N bit di memoria $\{a_i\}_{i=0}^{N-1}$

$$a = \sum_{i=0}^{N-1} a_i * 2^i$$

- Codifica i naturali tra 0 e 2^N-1

Caratteri

- char
- 8 bit
- Corrispondenza tra valori numerici e caratteri stabilita da tabella
- 0 non codifica caratteri utili, è usato come terminatore di stringhe (\0, NULL)

Interi con segno

- int
- N bit, rappresentazione complemento a 2
  - MSB ha peso negativo, gli altri bit peso positivo:

$$A = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i * 2^i$$

- Rappresenta numeri da -2^{N-1} a $2^{N-1}-1$

Complemento a 2

- Unifica il trattamento di somme e sottrazioni
  - ALU più semplice
- Per ottenere l'opposto di un numero si complementano i bit uno ad uno e si somma 1, ignorando overflow
  - Esercizio: calcolare il complemento di 0

Trucco (per umani)

- Da LSB verso MSB, si copiano i bit fino al primo 1 (compreso), quindi si complementano gli altri bit:
  - il complemento di
    0011 1100 è
    1100 0100
  - dal punto di vista circuitale non è più veloce

Overflow

- In generale, in un'operazione di somma, possiamo avere un overflow
  - nel caso di *unsigned int* l'overflow si manifesta con presenza di riporto nella somma degli MSB
  - nel caso di complemento a 2 si ha overflow solo se gli addendi hanno lo stesso segno
    - overflow sse il risultato della somma ha segno diverso da quello degli operandi

Virgola mobile

- Un numero reale può essere rappresentato come:

$$s \cdot m \cdot B^c$$

  - $s = \pm 1$ è il segno
  - $m \in \Re$ m>0 è la mantissa
  - c intero con segno e la caratteristica
  - B è la base di rappresentazione

- La rappresentazione è in *forma normale* se la mantissa soddisfa:
  - $1/B \leq m < 1$
- Tale rappresentazione si usa per i valori di tipo float
- Nello standard IEEE 754 si usano 32 bit:
  - 1 di segno (0 per positivi e 1 per negativi), 8 per caratteristica e 23 per la mantissa

Rappresentazione della caratteristica

- Si usa la forma polarizzata: c - (2^7-1) (offset di 127)
- la caratteristica assume valori tra -126 e 127 (config. estreme sono riservate)⇒ il massimo ordine di grandezza di un float è $2^{127} \approx 1000^{13}$

Rappresentazione della mantissa

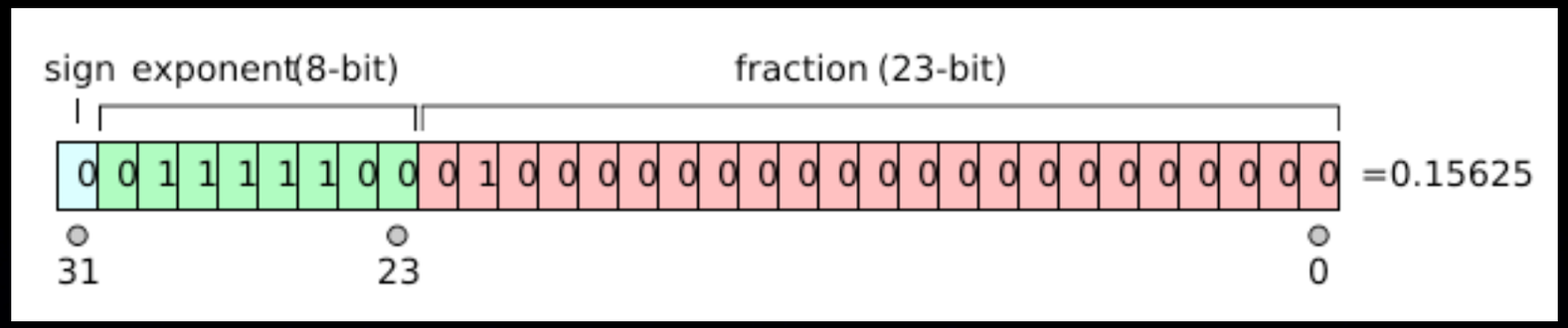
- La mantissa è una parte frazionaria

$$m = \sum_{k=1}^{\infty} m_k 2^{-k}$$

- Dato che è in forma normale l'MSB e sempre 1: se non fosse così m sarebbe < 1/2 dato che

$$\sum_{k=2}^{N} 2^{-k} < 1/2$$

- per cui si codificano i bit da m_1 a m_{24}



- s = 0 ⇒ +1

- esponente = 124 ⇒ c = -3

- mantissa = 1.01 (1 è m_1), $[1.01]_2 = [1.25]_{10}$

- + 1.25 * 2^{-3} = 0.15625

Errore di rappresentazione

- Per ogni ordine di grandezza rappresentato dalla caratteristica si hanno 2^{23} valori distinti
- la densità della copertura dei razionali non è omogenea ⇒ è più conveniente

  rappresentare l'errore in modo relativo

- Dato il valore razionale x e la sua approssimazione nell'insieme dei float, l'errore relativo è

$$\epsilon_r = \frac{x - \bar{x}}{x} = \frac{sm2^c - s\bar{m}2^c}{sm2^c} = \frac{m - \bar{m}}{m}$$

- la stima è invariante rispetto all'ordine di grandezza di x

- Considerando

$$m = 2^{-1} + \sum_{k=2}^{\infty} m_k 2^{-k}$$

$$\bar{m} = 2^{-1} + \sum_{k=2}^{24} m_k 2^{-k}$$

- si ottiene

$$\epsilon_r \leq \frac{2^{-24}}{\frac{1}{2}} = 2^{-23} \approx 10^{-7}$$

Doppia precisione

- Per ridurre l'errore si può usare il tipo *double*, rappresentato con 64 bit
  - segno: 1
  - caratteristica: 11
  - mantissa: 52
  - Errore relativo: 2^{-52}

Valori riservati della caratteristica

- c ed m tutti a 0 = 0
- c tutti ad 1 ed m tutti a 0 = ±∞
- c tutti ad 1 ed m≠0 = NaN

Errori di aritmetica in precisione finita

- Es. somma di float
- si allineano le caratteristiche dei due addendi al valore più alto
- si sommano le mantisse
- eventualmente si ri-normalizza la mantissa, facendo scorrere la caratteristica

$.465*10^{1}+.997*10^{3} = (.00465+.997)*10^{3} = 1.00165*10^{3} = .100165*10^{4}$

- Aumentare la dimensione della mantissa nella ALU non è un problema
- Può essere un problema il dover riportare il risultato nella rappresentazione esterna, es. troncando un numero di cifre che nel caso pessimo sono pari alla differenza tra le caratteristiche
- si rischia di ignorare completamente il valore dell'addendo minore !

Differenze tra int e float

- Hanno range diversi (banale)
- int: problema di overflow ma aritmetica esatta
- float: no overflow ma errori di approssimazione
- Gli *int* codificati con più di 25 bit non sono esattamente rappresentabili da *float*