NOTAZIONE BINARIA

NOTAZIONE POSIZIONALE

- Cifre uguali in **posizioni** diverse hanno significato diverso
- $4.34_{10} = (4 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2})_{10}$
- $a_{n-1}a_{n-2}...a_0 = \sum_{k=0}^{n-1} a_k b^k$
  - n cifre
  - b base
  - a_k cifra in posizione k

BASE BINARIA

- È la più semplice da manipolare per un computer
  - acceso/spentoSWDE4RF5T
  - tensione alta/bassa

CONVERSIONE DA BINARIO A DECIMALE

- $(1101)_2 = (1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 10^0)_{10} = (13)_{10}$

CONVERSIONE DA DECIMALE A BINARIO

- Parte intera elaborata in modo **indipendente** da quella frazionaria
- Conversione **parte intera**
  - $a_{k+1} = a_k \div 2$ (ed elimino eventuale resto)
  - se $a_k = 0$ mi fermo
  - risultato dato dalla serie dei resti iniziando dall'ultimo

```
100 / 2 = 50 resto 0
 50 / 2 = 25 resto 0
 25 / 2 = 12 resto 1
 12 / 2 =  6 resto 0
  6 / 2 =  3 resto 0
  3 / 2 =  1 resto 1
  1 / 2 =  0 resto 1
```

$(100)_{10} = (1100100)_2$

- Conversione **parte frazionaria**
  - se $a_k > 1$ -> $a_{k+1} = (a_k - 1) * 2$
  - se $a_k < 1$ -> $a_{k+1} = a_k * 2$
  - se parte decimale di $a_k = 0$ oppure ad un risultato ottenuto in precedenza -> mi fermo
  - risultato dato dalla serie delle parti intere partendo dal primo

```
0.35 ·2 = 0.7
0.7  ·2 = 1.4
0.4  ·2 = 0.8
0.8  ·2 = 1.6
0.6  ·2 = 1.2
0.2  ·2 = 0.4
```

$(0.35)_{10} = (0.01\overline{0110})_2$

RAPPRESENTAZIONE DI NUMERI NATURALI

- 8 bit -> $2^8 = 256$ disposizioni
  - $0_{10} = 0000\,0000_2$
  - $1_{10} = 0000\,0001_2$
  - $2_{10} = 0000\,0010_2$
  - ...
  - $255_{10} = 1111\,1111_2$

In generale

- rappresentazione n bit
- si possono rappresentare 2^n numeri interi
- $a \in [0, 2^n - 1] \cap \mathbb{N}$

RAPPRESENTAZIONE DI NUMERI INTERI RELATIVI

RAPPRESENTAZIONE CON MODULO E SEGNO

- Codifica
  - Primo **bit** (**+ significativo**): indica **segno**
    - 0 -> +
    - 1 -> -
  - Altri bit: indicano modulo del numero

- Esempio (8 bit)
  - $(0\,0001100)_2 = (+12)_{10}$
  - $(1\,0001100)_2 = (-12)_{10}$
- Due codifiche diverse per dire 0 (+0, -0) -> spreco
- Algoritmo di addizione -> troppo **lento** per una semplice somma

```
se (segno(A) == segno(B)):
        segno(S)=segno(A)
        |S|=(|A|+|B|)
altrimenti:
        se (|A| >= |B|):
                segno(S)=segno(A)
                |S|=|A|-|B|
        altrimenti:
                segno(S)=segno(B)
                |S|=|B|-|A|
```

In generale

- rappresentazione a n bit
- si possono rappresentare $2^n - 1$ numeri interi
- $a \in [-(2^{n-1} - 1), +(2^{n-1} - 1)] \cap \mathbb{Z}$

N.B.

- Cifre più significative -> posizione a sinistra
- Cifre meno significative -> posizione a destra

RAPPRESENTAZIONE CON COMPLEMENTO A DUE

- **Codifica**:
  - $a \geq 0$
    - rappresentazione: a
  - $a < 0$
    - rappresentazione: $(a + 2^n)$

- Esempio (8 bit)
  - $(0111\,1111)_2 = +127_{10}$
  - $(0000\,0001)_2 = +1_{10}$
  - $(0000\,0000)_2 = 0_{10}$
  - $(1111\,1111)_2 = 255_{10} => -1_{10}$
  - $(1000\,0000)_2 = 128_{10} => -128_{10}$
- Procedimento per **conversione** da **decimale negativo a binario**
  - 1. modo
    - sommo al numero 2^n e poi converto
  - 2. modo
    - codifico il modulo
    - scambio 0 con 1
    - sommo 1 al risultato
- Proprietà
  - **bit** più **significativo** rappresenta **segno**
  - solo **UNA** codifica per lo zero
- Addizione: si esegue **addizione binaria**
  - bisogna ignorare eventuale $(n+1)_{esima}$ cifra

• 0000 0101 +	+5 +
• 0000 0010	+2
• 0000 0111	+7

0000 0101 +	+5 +
1111 1110	-2
1|0000 0011	+3

In generale

- rappresentazione a n bit
- si possono rappresentare 2^n numeri interi
- $a \in [-(2^{n-1}), +(2^{n-1}-1)] \cap \mathbb{Z}$

ERRORE DI OVERFLOW

- Il numero è **più grande** della **codifica consentita**

OVERFLOW IN ADDIZIONE BINARIA

- Avviene se: $a + b \notin [-2^{n-1}, +2^{n-1} - 1] \cap \mathbb{Z}$
- Si può determinare analizzando le due cifre più significative del risultato
  - si ha **riporto** in **una sola** delle due cifre più significative

- 0111 1111 +127+
- 0000 0001 1
- 1000 0000 -128 !!

1000 0000 -128
1111 1111 -1
1|0111 1111 +127 !!

RAPPRESENTAZIONE IN VIRGOLA FISSA

- Il **separatore** si trova sempre nella **stessa posizione** rispetto alla sequenza dei bit

NUMERI REALI IN VIRGOLA MOBILE

- Notazione a **mantissa ed esponente**
- $1024.3 = 1.0243 * 10^3$
  - alcuni bit usati per la base (mantissa)
  - alcuni bit usati per l'esponente
- Computer utilizzano base 2
  - $(2^0 + 2^{-1} + 2^{-2} + ... + 2^{-23}) * 2^e = 1.110 ... 1 * 2^E$

IEEE 754

- Standard internazionale
- **Precisione singola** (32 bit)
  - 1 bit: segno
  - 8 bit: esponente con bias $e + 127$
  - 23 bit: mantissa normalizzata (senza 1 iniziale)
- **Precisione doppia** (64 bit)
  - 1 bit: segno
  - 11 bit: esponente con bias $e + 1023$
  - 52 bit: mantissa normalizzata (senza 1 iniziale)
- Numeri "**riservati**"
  - **0**
    - mantissa: 0
    - esponente: -127
  - **infinito**
    - mantissa = 0
    - esponente: +128
  - **NaN**
    - mantissa ≠ 0
    - esponente: +128

- In codifica a 32 bit
  - Numero più piccolo: $1.8 * 10^{-38}$
  - Numero più grande: $3.4 * 10^{+38}$

DENSITÀ DI NUMERI IN VIRGOLA MOBILE

- Si può calcolare con:
  - $\delta = 2^{-m} * 2^E$
    - m = bit mantissa
    - E = esponente
- Distanza tra due numeri reali rappresentabili a 32 bit: $\delta = 2^{-23} * 2^E$
  - 2^{-23} = numero più piccolo mantissa a 23 bit
  - E = valore esponente
  - la distanza dipende dal valore dell'esponente

ARROTONDAMENTO IN VIRGOLA MOBILE

- Alcuni numeri decimali non hanno una rappresentazione esatta in binario
  - Es. 4.35
- Può causare errori nelle somme

RAPPRESENTAZIONE ESADECIMALE

- Rappresentazione in **base 16**
  - $A = 10_{10}$
  - $B = 11_{10}$
  - $C = 12_{10}$
  - $D = 13_{10}$
  - $E = 14_{10}$
  - $F = 15_{10}$
- Conversione da binaria a esadecimale
  - $16 = 2^4$ -> raggruppo bit 4 a 4 **da destra**
- Esempio:
  - $0111\,1111_2 = 7F_{16} = 0x7F$

RAPPRESENTAZIONE OTTALE

- Rappresentazione in **base 8**
- Conversione da binaria a ottale
  - $8 = 2^3$ -> raggruppo bit 3 a 3 **da destra**
- Esempio
  - $100\,010_2 = 42_8$

RAPPRESENTAZIONE DI CARATTERI

- A ciascun carattere viene associato un numero naturale
- **ASCII** (American Standard Code for Information) - 7 bit
    - 128 caratter
        - alfabeto americano + numeri e simboli
- **ASCII esteso** - 8 bit
    - 256 caratteri
    - codifica tutti i caratteri dell'alfabeto occidentale
- **UNICODE** - 16bit
    - tutti i caratteri per tutte le lingue
    - i primi 7 bit corrispondono alla codifica ASCII