RAPPRESENTAZIONE DELL'INFORMAZIONE

Nei calcolatori elettronici le informazioni sono salvate e **RAPPRESENTATE** in codifica **BINARIA**, cioè come sequenze di bit, è l'unità di misura elementare dell'informazione (minima) è il **bit** (binary digit) ⇒ 1/0 (perché più semplice in elettronica)

Il calcolatore è dotato di dispositivi di INPUT e OUTPUT
I dispositivi che traducono le informazioni in codice binario sono i TRASDUTTORI.

I caratteri sono rappresentati con la codifica **ASCII**; essa lavora su 8 bit (1 byte (informazione minima utilizzata nel calcolatore)) da cui rappresentiamo 256 caratteri.
Infatti con m bit rappresentiamo:

2^m caratteri

Per la rappresentazione dei NUMERI si divide in naturali, reali e interi. Tale rappresentazione è sempre FINITA (per tutti i numeri abbiamo m bit ⇒ INTERVALLO DI RAPPRESENTABILITÀ)

- **NATURALI** (0, 1, 2, ...)

In base 10 ogni cifra ha un peso a seconda della posizione che occupa (potenza del 10):

$123 \Rightarrow 1\cdot10^2 + 2\cdot10^1 + 3\cdot10^0$ (123 → 10^2, 10^1, 10^0)

CONVERSIONE $b_2 \Rightarrow b_{10}$

Nel calcolatore abbiamo ancora una rappresentazione POSIZIONALE ma in base 2 cioè i bit assumono valore a seconda delle potenze del 2:

1001 → 2^3, 2^2, 2^1, 2^0

Con 4 bit l'intervallo di rapp è [0,15] cioè:

da 0 a $2^m - 1$ (sempre 2^m configurazioni) ||

CONVERS. $b_{10} \Rightarrow b_2$

Per la conversione da base 10 a base 2 si utilizza l'algoritmo MOD/DIV (and); si lavora eseguendo divisioni successive per 2 e calcolando i resti (in ordine inverso)

$123 : 2 = 61 + 1$ ← cifra peso 2^0
$61 : 2 = 30 + 1$ ← cifra peso 2^1
$30 : 2 = 15 + 0$ ← cifra peso 2^2
$\vdots$
$1 : 2 = 0 + 1$ ← cifra peso 2^6

...fino a raggiungere quoziente 0

1111011

CONV. $b_{10} \Rightarrow b_\beta$

Tale procedimento in realtà è utilizzato per la conversione da base 10 a base β (BETA).

CONV. $b_\beta \Rightarrow b_\alpha$

Per la conversione da base β a base α quindi si converte $\beta \Rightarrow$ base$_{10} \Rightarrow \alpha$. Tale conversione si semplifica se la conversione avviene tra base 2 e base potenza del 2: i bit infatti vengono raggruppati in gruppi pari all'esponente della base potenza del 2.

CONV. $b_2 \Rightarrow b_{2^m}$

base 2: 01 11 00 11
base $4 = 2^2$: 1 3 0 3

conv. $b_2^m \Rightarrow b_2$

Per la conversione da base potenza del 2 a base 2 si esegue il procedimento inverso, cioè invece di raggruppare si **espande** ogni cifra in sequenza di bit.

Con la codifica binaria è possibile, essendo una rappresentazione posizionale, eseguire le operazioni aritmetiche come se si trattasse di una codifica decimale:

$$\begin{array}{r} 5_{+} \\ 3 \end{array} \Rightarrow \begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 0101_2 \\ 0011 \\ \hline 1000 \end{array}$$

L'operazione di somma funziona quindi come nell'operazione con numeri decimali e quindi con l'uso del riporto.

La condizione di **NON RAPPRESENTABILITÀ** si ha nel caso in cui il riporto sia presente in una posizione superiore al bit più significativo possibile(aldilà del numero di bit rappresentabili). Si parla di **OVERFLOW**

$$\begin{array}{r} {\scriptstyle 1} \\ 0111_2 \\ 1011 \\ \hline 10010 \end{array}$$

segnalato nel registro di stato con il bit di overflow

www.master copy corsi.com 085/8312126 388/1983745

PROGRAMMA IN DEV-C++

```
#include <iostream>          // libreria funzioni ingresso-uscita (se molto <iostream.h> non c'è bisogno di using namespace...
#include <cstdlib>           // "system()" (facoltativa): è la libreria che impartisce comandi al Sist. Oper.
using namespace std;         // nel nostro programma possiamo così usare "nomi" standard utilizzati per la programmaz.
                             (ENTRY POINT PROGRAMMA)
int main()                   vuol dire: VAI A CAPO E MOSTRA A VIDEO IMMEDIATAMENTE
{ cout << "Ciao mondo!" << endl;   // COUT: è il nome del video (grazie alle librerie)   << implementa un'operazione d'uscita (a video)
  system("pause");           // attende l'immissione di carattere da tastiera
  return 0;                  // restituisce 0 al programma (o Sist. Oper.) e indica che è andato bene
}                            // le parentesi racchiudono le istruzioni;
```

'\t' '\n'

Scaricato un ambiente di programmazione viene scaricato:

- LIBRERIE: consistono di un insieme di funzioni predefinite che permettono operazioni varie (se incluse nel programma)
- COMPILATORE: è lo strumento che traduce il linguaggio di programmazione in ling. macchina.
- LINKER: collega il codice del programma compilato con il codice delle librerie usate creando così il file ESEGUIBILE

ERRORI DI SINTASSI — ERRORI DI LINKAGGIO (possono dipendere dall'installazione non corretta)

LAVORANDO A LINEA DI COMANDO DOS:

File sorgente ⇒ NOMEFILE.cpp

COMPILA: g++ [-c] NOMEFILE.cpp -o NOMEFILE.O — è l'istruzione che richiama il compilatore (G++) e quindi compila il sorgente e genera il file OGGETTO (MODULO)

con compila / senza linka

LINKA: g++ NOMEFILE.O -o PROGETTO.EXE [-l"lib"] — richiama il linker (G++) generando il file eseguibile

eventuali librerie

DEBUG:

- step a step
- con break-point
- fino alla posizione del cursore

Per abilitare il DEBUG: Strumenti → Opzioni di compilazione → Generazione di Codice → Linker → Generare le info per il debug (YES)

SPUNTA "Aggiungi i comandi..." e poi -g3 -Wall

Un alternativa alla if è il seguente (l'operatore ternario ?:).

(COND) ? ESPRES1 : ESPRES2 — sono solo un ESPRESSIONE (non di più).

se è vero / se è falso

NUMERI INTERI

Gli interi hanno diverse rappresentazioni (esattamente tre se si considera la traslazione):

- **MODULO E SEGNO**: il modulo è il valore assoluto del numero. In tale rappres. si usa il bit più significativo per il segno; e ciò significa:

  0 ⇒ +
  1 ⇒ −

  Gli altri bit sono usati per codificare il modulo:

  4 bit:
  −3 ⇒ 1011

  Quindi il valore del numero sarà:

  $$\begin{cases} \text{se } (a_{p-1} == 0) & m = +A \\ \text{else} & m = -(A - 2^{p-1}) \end{cases}$$

  (bit più significativo; intera stringa; ↳ valore bit più significativo sottratto all'intera stringa)

  Lo zero sarà quindi rappresentato:

  0 000 ⇒ +0
  1 000 ⇒ −0

  Vi è una DOPPIA RAPPRESENTAZIONE DELLO ZERO

  L'intervallo di rappresentabilità sarà dato quindi (bit−1 sono usati per il modulo):

  $$\pm\left(2^{bit-1} - 1\right)$$

  da $\left[-\left(2^{bit-1} - 1\right), +\left(2^{bit-1} - 1\right)\right]$ È SIMMETRICO!

  Potrei anche vederlo in decimale: se supera l'intervallo vuol dire che è negativo.

- **COMPLEMENTO ALLA BASE (o A 2)**: numeri positivi e zero con bit più significativo 0, negativi con 1.

  $$\begin{cases} \text{se } (a_{p-1} == 0) & m = +A \\ \text{else} & m = -(\text{N}A + 1) \end{cases}$$

  (il complemento (cioè cambio tutti i bit) o negato)

  0000 0
  0001 1
  ↓
  0111 7
  1000 −8
  1001 −7
  ↓
  1111 −1

  7 = 111 → −1 ⇒ 110 N 001

  1001 = −(6+1) = −7
  ↳ complemento
  0110

  Sparisce il doppio zero: 0000 = 0 / 1000 = −8 e quindi aumenta l'intervallo di rappresentabilità (che diventa asimmetrico).

  $$\left[-\left(2^{bit-1}\right), +\left(2^{bit-1} - 1\right)\right] \qquad 4\,bit \Rightarrow [-8, +7]$$

  Il vantaggio di tale notazione è che consente di eseguire le operazioni aritmetiche con gli stessi circuiti utilizzati per i naturali (hardware più semplice). Ciò è dovuto al fatto che il negativo più piccolo (−8) ha tutti 0.

  Particolare è però il caso dell'overflow:

  $$\begin{array}{r} {\scriptstyle 1\ \ 1\ \ \ \ } \\ 0111 + \\ 1111 \\ \hline 1\ 0110 \end{array}$$

  7 + (−1) = 6 produce un riporto a 1 ma le restanti cifre producono proprio 6. Non si ha l'overflow perché abbiamo positivi e negativi.

  FLAG OVERFLOW = 0
  CARRY FLAG (del riporto) = 1

In generale non si ha l'overflow se hanno segno diverso. Se sommo numeri di uguale segno ho l'overflow se il risultato cambia di segno

7+5 → 0111 + 0101 = 1100 (CAMBIA DI SEGNO) — CARRY FLAG=0, FLAG OVERFLOW = 1

NUMERI "REALI"

Per i numeri reali ci sono due rappresentazioni. Trattandosi comunque di numeri infinitesimi, nell'elaboratore (trattandosi di rappresentazione finita) avremo una certa approssimazione.

- **REGOLA VIRGOLA FISSA** = si stabilisce quanti bit usare per la parte intera e quanti per quella frazionaria (parte segno). La parte intera rappresenta sempre l'intervallo di rappresentabilità; la parte frazionaria ci fornisce la precisione ("densità") della parte decimale.

56,21 — si tratta sempre di una codifica posizionale

10^1 10^0 10^{-1} 10^{-2}

MASTER COPY

Tel. 388/9837745

+1110.01 ⇒ +14.25

La parte intera si rappresenta con le tecniche note.

Facendo la conversione inversa: 331,6875

331 | 2 → 1 | 165 | 2

- basta convertire con MOD/DIV per la parte intera (perché cerchiamo potenze positive del due).
- Per la parte frazionaria (trattandosi di potenze negative) si segue l'algoritmo opposto: si moltiplica per 2 e si conserva la parte intera (in ordine normale).

0.6875 · 2 = 1.375
0.375 · 2 = 0.750
0.75 · 2 = 1.5
0.5 · 2 = 1.0

101001011.1011

Termina quando il prodotto è 1 e parte frazionaria pari a 0 (o quando si è raggiunta la precisione desiderata).

Se eseguo la somma delle potenze $2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}$ ... ottengo la parte frazionaria.
Ci possono essere però casi in cui la parte frazionaria finita in base 10 diventa periodica in base 2. Ciò significa che a volte il numero memorizzato non è proprio uguale nella macchina ma sarà memorizzato con una certa approssimazione (ERRORE DI TRONCAMENTO).

2,3

0.3 · 2 = 0.6
0.6 · 2 = 1.2
0.2 · 2 = 0.4
0.4 · 2 = 0.8
0.8 · 2 = 1.6
0.6 · 2 = 1.2

1.0100110