La Rappresentazione dell'Informazione e la Codifica dei Numeri

Informatica A, AA23/24

15 Settembre 2023

Giacomo Boracchi

https://boracchi.faculty.polimi.it/

giacomo.boracchi@polimi.it

Cos’è l’Informatica?

Scienza della rappresentazione e dell’elaborazione dell’informazione.

Scienza: ovvero una **conoscenza sistematica** e **rigorosa** di tecniche e metodi.

Informazione: l’oggetto dell’investigazione scientifica (informazione intesa come entità astratta e come tecnologie per la sua gestione)

Rappresentazione: il modo in cui l’informazione viene strutturata e trasformata in dati fruibili da macchine

Elaborazione: uso e trasformazione dell’informazione per un dato scopo. L’elaborazione deve poter essere eseguita da **macchine** che processano dati.

[da «Informatica Arte e Mestiere»]

Rappresentazione dei Numeri

Codifica dei Numeri in Base 10

Le cifre che abbiamo a disposizione sono 10

$$A_{10} = \{0, 1, \dots, 9\}$$

Utilizziamo una **codifica posizionale**, quindi le cifre in posizioni differenti hanno un significato differente

- Es numero di 4 cifre
  - $\mathbf{3401} = \mathbf{3} \times 10^3 + \mathbf{4} \times 10^2 + \mathbf{0} \times 10^1 + \mathbf{1} \times 10^0$

Con m cifre posso rappresentare 10^m numeri distinti:

$$0, \dots, 10^m - 1$$

Codifica dei Numeri in una Base Qualsiasi

Ogni codifica ha un insieme di cifre (dizionario) A

- In base 10, il dizionario è $A_{10} = \{0, \dots, 9\}$

Un numero è una sequenza di m cifre

$$a_{m-1} \dots a_1 a_0 \text{ con } a_i \in A$$

- 8522 è una sequenza di 4 cifre di A_{10}, $\{8,5,2\} \subset A_{10}$.

Manteniamo un **codifica posizionale**: ogni cifra assume un significato diverso in base alla sua posizione nel numero.

- a_n è la cifra **più significativa**
- a_0 è la cifra **meno significativa**

Es: in **8**52**2**, **8** è la cifra più significativa, **2** quella meno.

8522 è diverso da 2852, 8252,… che pur contengono le stesse cifre

Codifica dei Numeri: Notazione Posizionale

Dato un numero N_{10}, in base 10 contenente m cifre scritto come $a_{m-1}a_{m-2} \ldots a_1 a_0$ questo corrisponde a:

$$N_{10} = a_{m-1} \times 10^{m-1} + a_{m-2} \times 10^{m-2} + \ldots + a_0 \times 10^0$$

$$(a_{m-1}a_{m-2} \ldots a_1 a_0)_{10} = \sum_{i=0}^{m-1} a_i \times 10^i \, , \qquad a_i \in A_{10}$$

Es: $(8522)_{10} = 8 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 2 \times 10^0$

Con m cifre in A_{10} quanti numeri posso esprimere? 10^m

Considerando gli interi positivi, posso scrivere tutti numeri tra $[0, 10^m - 1]$

- *Es:* $m = 1$ copro $[0, 10 - 1]$ (cioè $[0,9]$ i.e., A_{10})

  $m = 3$ copro $[0, 10^3 - 1]$ (cioè $[0, 999]$)

Rappresentazioni Posizionali in Base p

Consideriamo **rappresentazioni posizionali in base $\boldsymbol{p}$** (con $p > 0$) e chiamiamo A_p il dizionario di p cifre:

- se $p \leq 10$ prendiamo le cifre di A_{10}, $A_p = \{0, \dots, p-1\}$
- se $p > 10$ aggiungiamo simboli $A_p = \{0, .., 9, A, B..\}$

Un numero di m cifre in base p:

$$N_p = a_{m-1} \times p^{m-1} + a_{m-2} \times p^{m-2} + .. + a_0 \times p^0$$

$$N_p = a_{m-1}a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times \boxed{p^i}, \qquad \boxed{a_i \in A_p}$$

Con m cifre in A_p quanti numeri posso esprimere: p^m

Considerando gli interi positivi, posso scrivere tutti numeri tra $[0, p^m - 1]$

Codifica dei numeri in base p: Esempi

Es: $m = 1$ e $p = 7$, copro $[0, 7 \; - 1]$ (cioè [0,6])

$m = 4$ e $p = 7$, copro $[0, 7^4 \; - 1]$ (cioè [0,2400])

$m = 1$ e $p = 13$, copro $[0, 13 \; - 1]$ (cioè [0,12])

$m = 4$ e $p = 13$, copro $[0, 13^4 \; - 1]$ (cioè [0,28560])

Al crescere di p cresce il «potere espressivo» del dizionario (con lo stesso numero di cifre posso scrivere molti più numeri).

Le targhe

Quante diverse targhe italiane ci possono essere (ignoriamo targhe speciali come CC , CRI, EI,..)

Lettere in uso: 24 (cioè 26 – 2, I e O non vengono utilizzate)

Le targhe

Quante diverse targhe italiane ci possono essere (ignoriamo targhe speciali come CC , CRI, EI,..)

Lettere in uso: 24 (cioè 26 – 2, I e O non vengono utilizzate)

Nro di targe : $24^4 * 10^3 = 331776000$

Se usassi numeri in base 10: 9 cifre

Se usassi numeri in base 2 $\approx \log_2(331776000) = 23$ cifre

Codifica dei Numeri in Base 2

I calcolatori sono in grado di operare con informazioni **binarie**. Quindi $p = 2$ e $A_2 = \{0, 1\}$

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + .. + a_0 \times 2^0$$

$$N_2 = a_{m-1}a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times 2^i \ , \qquad a_i \in \{0,1\}$$

Un bit (*binary digit*) assume valore 0/1 corrispondente ad un determinato *stato fisico* (alta o bassa tensione nella cella di memoria)

Con m bit posso scrivere 2^m numeri diversi, ad esempio tutti gli interi nell'intervallo $[0, 2^m - 1]$

Il byte è una sequenza di 8 bit ed esprime $2^8 = 256$ numeri diversi (ad esempio gli interi in $[0,255]$)

00000000, 00000001, 00000010, …, 11111111

Altre Codifiche che consideriamo

Codifica **ottale** (in **base 8**)

- $A_8 = \{0, 1, \ldots, 7\}$
- con m cifre in A_8 scrivo i numeri da $[0, 8^m - 1]$

Codifica **esadecimale**, (in **base 16**)

- $A_{16} = \{0, 1, \ldots, 9, A, B, C, D, E, F\}$,
- Per le conversioni $A = 10$, …, $F = 15$.
- con m cifre in A_{16} scrivo i numeri da $[0{,}16^m - 1]$.

Codifica dei numeri in base 2: Esempi

Quante combinazioni di m elementi posso realizzare se ogni elemento lo scelgo tra 2 diversi?

Codifica dei numeri in base 2: Esempi

Quante combinazioni di m elementi posso realizzare se ogni elemento lo scelgo tra 2 diversi?

$$2^m$$

Codifica dei numeri in base 2: Esempi

Quante combinazioni di m elementi posso realizzare se ogni elemento lo scelgo tra 2 diversi?

$$2^m$$

Il massimo intero positivo rappresentabile con m bit?

Codifica dei numeri in base 2: Esempi

Quante combinazioni di m elementi posso realizzare se ogni elemento lo scelgo tra 2 diversi?

$$2^m$$

Il massimo intero positivo rappresentabile con m bit?

$$2^m - 1$$

… ricordarsi di partire da 0 !

Codifica dei numeri in base 2: Esempi

Ad esempio:

Con 1 cifra in base 2, copro $[0, 2^1 - 1]$ (cioè $[0, 1]$)

Con 4 cifre in base 2, copro $[0, 2^4 - 1]$ (cioè $[0, 15]$)

Con 8 cifre in base 2, copro $[0, 2^8 - 1]$ (cioè $[0, 255]$)

Con 16 cifre in base 2, copro $[0, 2^{16} - 1]$ (cioè $[0, 65535]$)

In binario ho bisogno di molte più cifre rispetto al decimale per esprimere gli stessi numeri

… e tutte le altre potenze di 2?

È necessario imparare le potenze di 2!

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

E i loro legami con l'informatica:

- Byte = 8 bit
- KiloByte (kB) = 10^3 Byte
- MegaByte (MB) = 10^6 Byte
- GigaByte (GB) = 10^9 Byte
- TheraByte (TB) = 10^{12} Byte

Convertire in base 10

da base 2 (o altre basi) a base 10

Conversione Binario-Decimale

Utilizziamo la definizione di numero in notazione posizionale

$$N_2 = a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + .. + a_0 \times 2^0$$

Es.

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$$

$$(1100010)_2 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2 = (98)_{10}$$

Osservazioni

In binario i numeri che terminano con 1 sono dispari, quelli con 0 sono pari.

- L'unico modo per avere un numero dispari nella somma è aggiungere $2^0 = 1$

Le conversioni di numeri con bit tutti a 1 si calcolano facilmente

$$(111111)_2 = (1000000)_2 - (1)_2 = 2^6 - 1 =$$

Conversioni ottale/esadecimale ➡ decimale

È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times 16^i \ , \qquad a_i \in A_{16}$$

Conversioni ottale/esadecimale ➡ decimale

È possibile utilizzare le definizioni precedenti per convertire da ottale/esadecimale in base 10

$$N_{16} = a_{m-1}a_{m-2} \dots a_1 a_0 = \sum_{i=0}^{m-1} a_i \times 16^i \,, \qquad a_i \in A_{16}$$

Es:

$$(31)_8 = 3 * 8 + 1 = (25)_{10}$$

$$(A170)_{16} = A * 16^3 + 1 * 16^2 + 7 * 16$$

$$= 10 * 4096 + 1 * 256 + 7 * 16 = (41328)_{10}$$

$$(623)_8 = 6 * 8^2 + 2 * 8 + 3 * 8^0 = 6 * 64 + 16 + 3 = (403)_{10}$$

$$(623)_{16} = 6 * 16^2 + 2 * 16 + 3 * 16^0 = 6 * 256 + 32 + 3 = (1571)_{10}$$

Convertire in base 2

da base 10 (o altre basi) a base 2

Conversione Decimale ➡ Binario

Metodo delle divisioni successive:

Per convertire 531 opero come segue:

- 531 / 2 = 265 + 1
- 265 / 2 = 132 + 1
- 132 / 2 = 66 + 0
- 66 / 2 = 33 + 0
- 33 / 2 = 16 + 1
- 16 / 2 = 8 + 0
- 8 / 2 = 4 + 0
- 4 / 2 = 2 + 0
- 2 / 2 = 1 + 0
- 1 / 2 = 0 + 1

Divisione intera tra il numero e 2

Il risultato della divisione precedente viene successivamente diviso

Si continua fino a quando il risultato della divisione non diventa 0 (e considero comunque il resto!)

Conversione Decimale ➡ Binario

Metodo delle divisioni successive:

Per convertire 531 opero come segue:

- 531 / 2 = 265 + 1
- 265 / 2 = 132 + 1
- 132 / 2 = 66 + 0
- 66 / 2 = 33 + 0
- 33 / 2 = 16 + 1
- 16 / 2 = 8 + 0
- 8 / 2 = 4 + 0
- 4 / 2 = 2 + 0
- 2 / 2 = 1 + 0
- 1 / 2 = 0 + 1

Cifra meno significativa

I resti della divisione intera, letti dall'ultimo al primo, identificano il numero binario

$$(531)_{10} = (1000010011)_2$$

Cifra più significativa

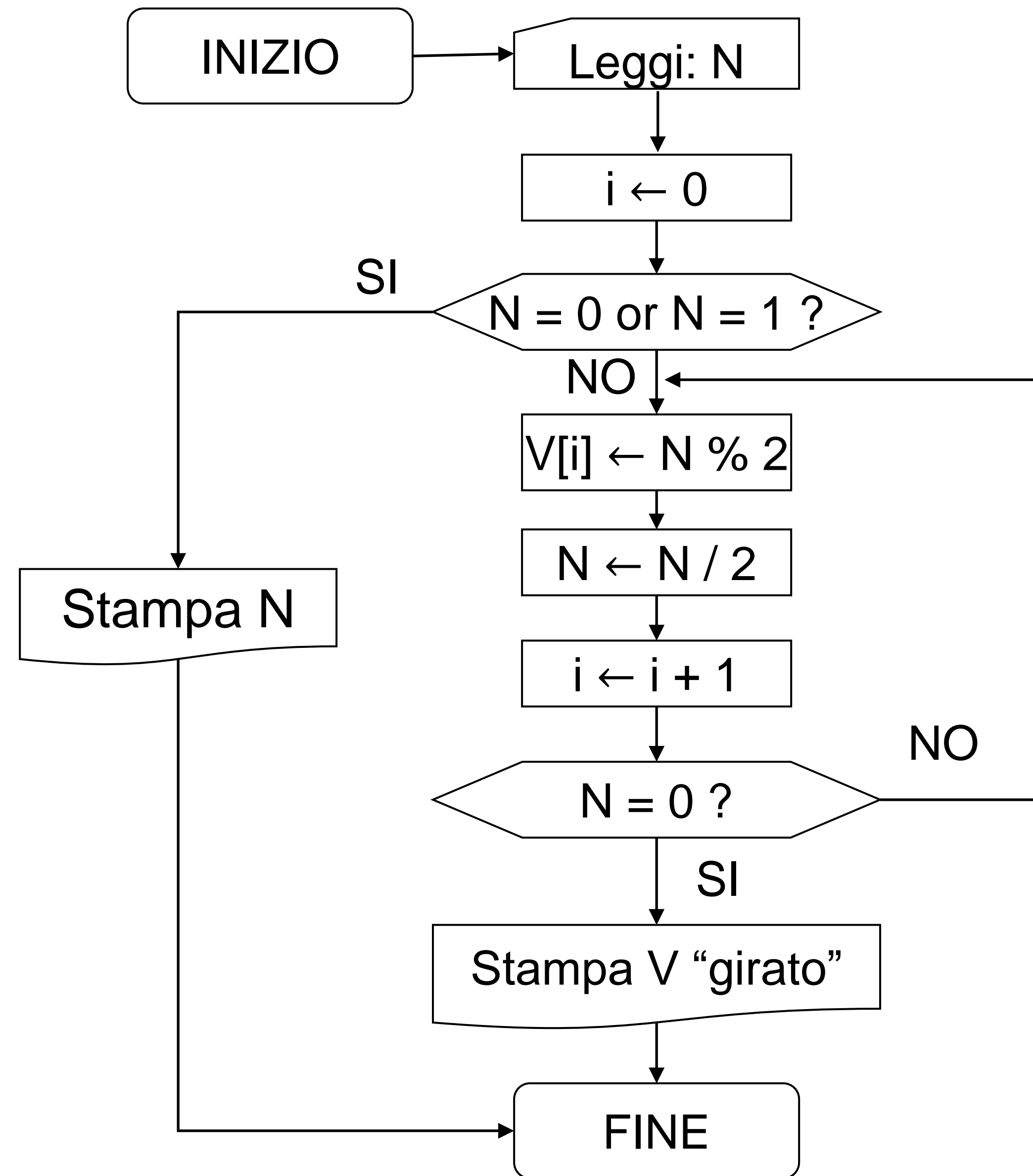
Algoritmo per convertire da base 10 a base 2

Assumiamo di scrivere il numero binario in un'opportuna struttura dati: un vettore!

V[i] è la posizione i-sima del vettore

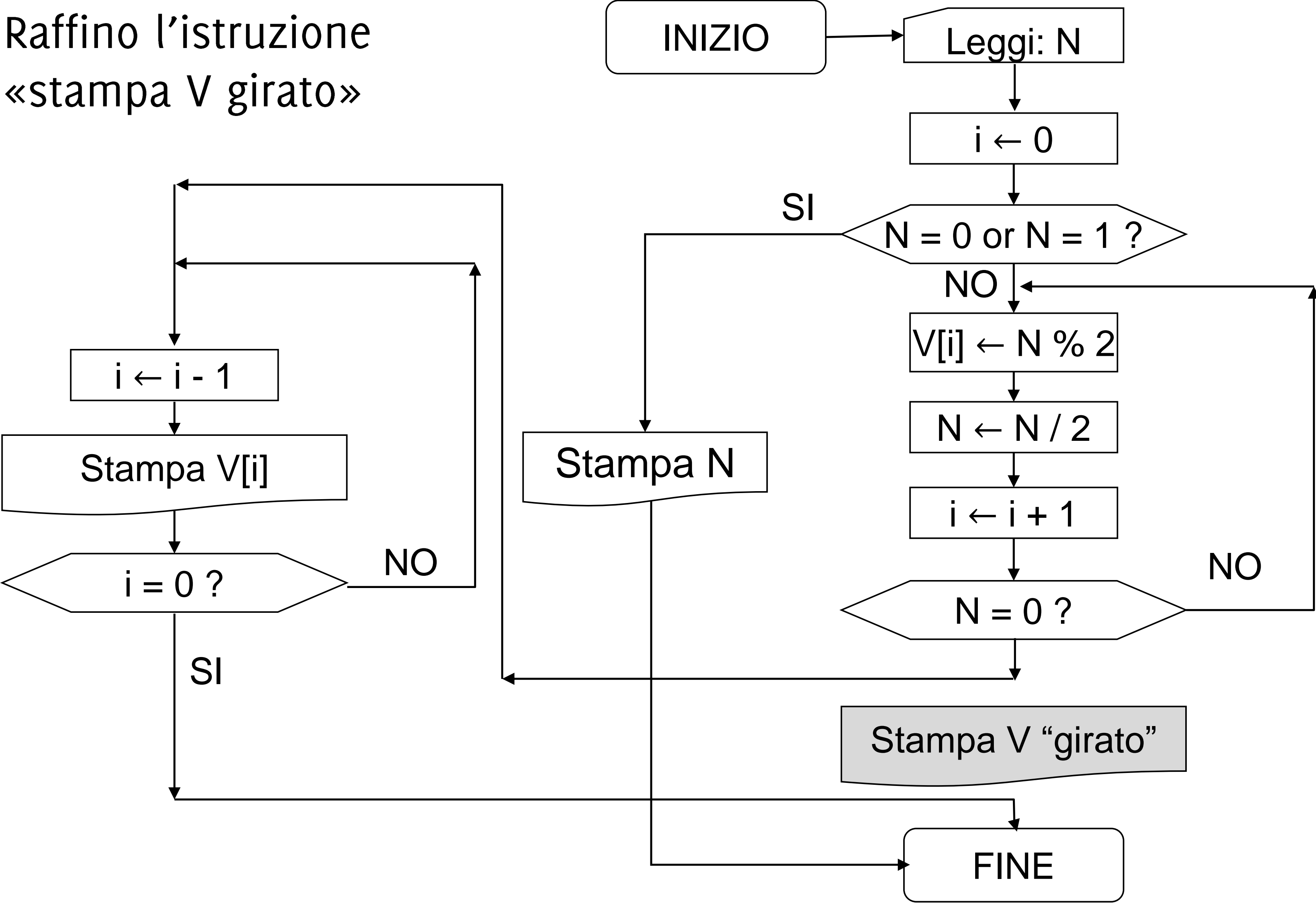
% calcola il resto della divisione intera

/ è la divisione intera



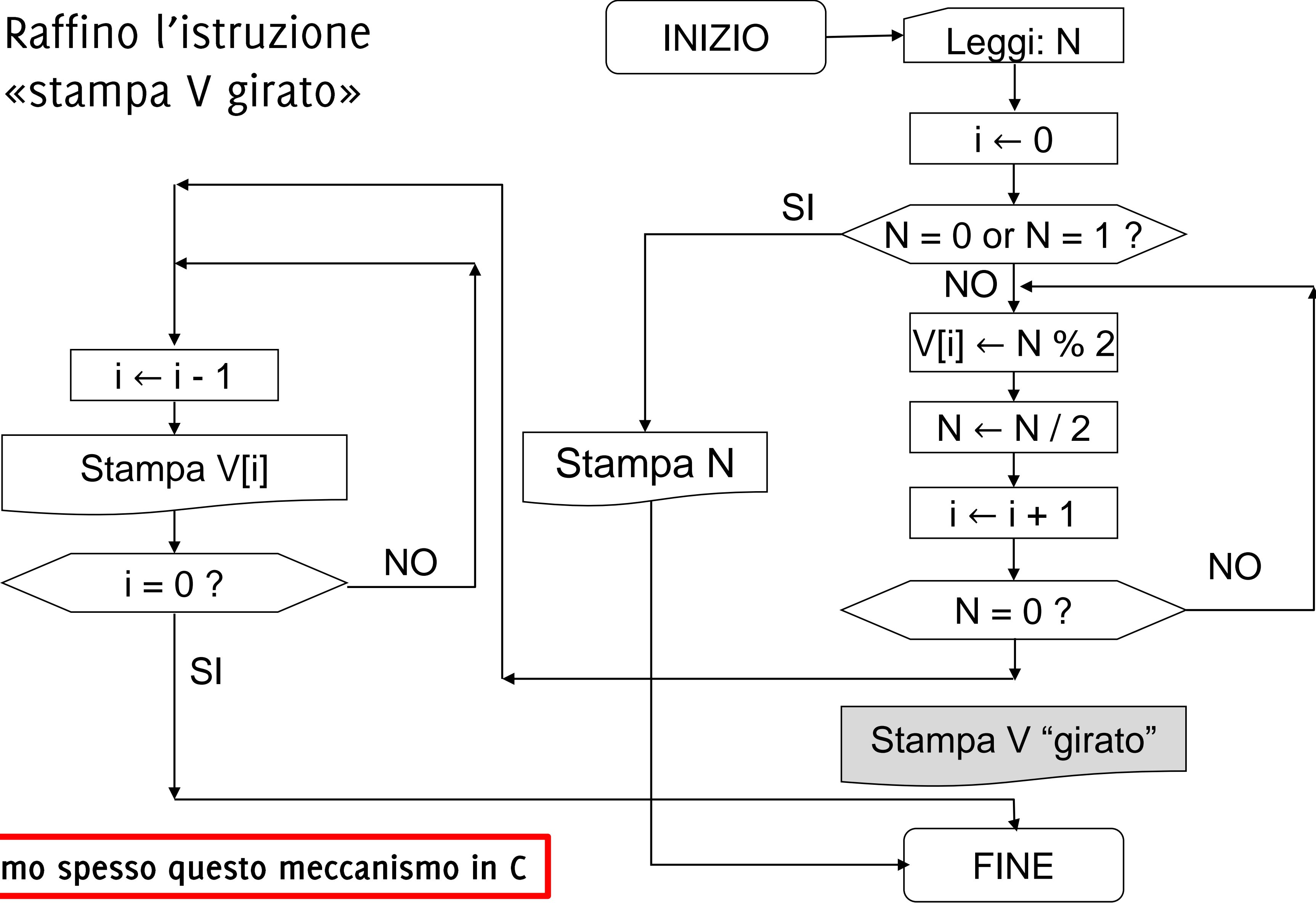
Algoritmo per convertire da base 10 a base 2

Raffino l'istruzione «stampa V girato»



Algoritmo per convertire da base 10 a base 2

Raffino l’istruzione «stampa V girato»



Vedremo spesso questo meccanismo in C

TODO: Conversione Decimale ➡ Binario

Scrivere un programma che esegue la conversione decimale binario e salva il risultato in un'opportuna struttura dati prima di visualizzarlo

Modificare il programma convertire da decimale ad una base p qualunque con $p \leq 16$ specificata dall'utente durante l'esecuzione del programma.

NB: l'algoritmo delle divisioni successive vale rispetto a qualunque base

NB: controllare che il numero inserito sia compatibile con il numero massimo di bit allocati

Conversioni ottale/esadecimale ➡ decimale

Convertire in base 2 i seguenti numeri

$(31)_8 \rightarrow 1\times 8^0 + 3\cdot 8^1 = (25)_{10}$

$(A170)_{16}$

$(623)_8$

$(623)_{16}$

25	1
12	0
6	0
3	1
1	1
0	

$2^4 \; 2^3 \; 2^0$

$(25)_{10} = (11001)_2$

$1 + 8 + 16$

Conversioni ottale/esadecimale ➡ decimale

Convertire in base 2 i seguenti numeri

$$(31)_8 = (25)_{10}$$

$$(A170)_{16} = (41328)_{10}$$

$$(623)_8 = (403)_{10}$$

$$(623)_{16} = (1571)_{10}$$

Conversioni ottale/esadecimale ➡ decimale

Convertire in base 2 i seguenti numeri

$$(31)_8 = (25)_{10} = (11001)_2$$

$$(A170)_{16} = (41328)_{10} = TBD$$

$$(623)_8 = (403)_{10} = (110010011)_2$$

$$(623)_{16} = (1571)_{10} = (11000100011)_2$$

Conversioni ottale/esadecimale ➡ binario

È possibile passare in base 10 e quindi utilizzare l'algoritmo delle divisioni successive

È tuttavia più comodo fare delle conversioni direttamente dalla rappresentazione binaria

Esprimere ogni sequenza di 3 numeri binari in base 8

- $(1231)_{10} = (10011001111)_2 = (010\ 011\ 001\ 111)_2$
- $(1231)_{10} = (2317)_8$

$(2 \quad 3 \quad 1 \quad 7)_8$

Esprimere ogni sequenza di 4 numeri binari in base 16

- $(1231)_{10} = (10011001111)_2 = (0100\ 1100\ 1111)_2$
- $(1231)_{10} = (4CF)_{16}$

$(4 \quad C \quad F)_{16}$

Queste tecniche possono essere usate per convertire da base decimale in esadecimale/ottale passando facilmente in binario

Convertire in base 2 i seguenti numeri

$$(31)_8 = (25)_{10} = (011\ 001)_2$$

$$(A170)_{16} \rightarrow (1010\ 0001\ 0111\ 0000)_2$$

$$(623)_8 = (403)_{10} = (110\ 010\ 011)_2$$

$$(623)_{16} = (1571)_{10} = (0110\ 0010\ 0011)_2$$

Ora è più facile convertire $A170$

Conversioni e rapide bin ➡ dec

In binario si definisce una *notazione abbreviata*, sulla falsariga del sistema metrico-decimale:

K	$= 2^{10}$	$= 1.024 \approx 10^3$	(Kilo)
M	$= 2^{20}$	$= 1.048.576 \approx 10^6$	(Mega)
G	$= 2^{30}$	$= 1.073.741.824 \approx 10^9$	(Giga)
T	$= 2^{40}$	$= 1.099.511.627.776 \approx 10^{12}$	(Tera)

È curioso (benché *non* sia casuale) come K, M, G e T in base 2 abbiano valori molto prossimi ai corrispondenti simboli del sistema metrico decimale, tipico delle scienze fisiche e dell'ingegneria

Fino a 2^{40} l'errore risulta < 10 % (infatti la seconda cifra è sempre 0)

Ma allora…

… diventa molto facile e quindi rapido **calcolare il valore decimale approssimato delle potenze di 2**, anche se hanno **esponente grande**

Infatti basta:

- Tenere a mente l'elenco dei valori esatti delle prime dieci potenze di 2 [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
- Scomporre in modo additivo l'esponente in contributi di valore 10, 20, 30 o 40, "leggendoli" come successioni di simboli K, M, G oppure T

Primo esempio

Tieni ben presente che:

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

E ora dimmi in un secondo (non di più) quanto vale, approssimativamente, 2^{17}

Primo esempio

Tieni ben presente che:

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

E ora dimmi in un secondo (non di più) quanto vale, approssimativamente, 2^{17}

- risposta: “128 mila”
- infatti $2^{17} = 2^{(7+10)} = 2^7 \times 2^{10} \approx 128\,K$
- in realtà, $2^{17} = 131.072$
- Errore percentuale $= 1 – 128.000/131.072 \approx 2{,}3\,\%$

Altri esempi

$2^{24} =$

$2^{35} =$

$2^{48} =$

$2^{52} =$

Altri esempi

$2^{24} = 2^4 * 2^{20} = 16$ M, leggi "16 milioni"

$2^{35} = 2^5 * 2^{30} = 32$ G, leggi "32 miliardi"

$2^{48} = 2^8 * 2^{40} = 256$ T, leggi "256 bilioni"

o anche $= 2^8 * 2^{10} * 2^{30} = 256$ K G, leggi "256 mila miliardi"

$2^{52} = 4\, K\, T$, leggi "4 mila bilioni",

o anche = 4 M G, leggi "4 milioni di miliardi"

N.B.: l'approssimazione è sempre per difetto

Al contrario… (dec → bin)

Si osservi come $10^3 = 1000 \approx 1024 = 2^{10}$,

con errore $= 1 - \frac{1000}{1024} \approx \ 2{,}3\ \%$

Pertanto, preso un intero n, si ha:

- $10^n = (10^3)^{n/3} \approx (2^{10})^{n/3} = 2^{10\,n/3}$

Dimmi subito quanto vale (circa) in base 2:

- 10^9, risposta: circa $2^{10\times 9/3} = 2^{30}$

  con errore: $1 - 2^{30}/10^9 \approx -7{,}3\ \%$ (approx. eccesso)
- 10^{10}, risposta: circa $2^{10\times 10/3} \approx 2^{33}$

  con errore: $1 - 2^{33}/10^{10} \approx 14{,}1\ \%$ (approx. difetto)

L'approssimazione è per eccesso o per difetto

Piccolo trucco aritmetico

Calcolare g^x in modo efficiente

Piccolo trucco aritmetico

Calcolare g^x in modo efficiente utilizzando la rappresentazione binaria di x ed il fatto che $g^{2x} = g^x * g^x$

Si consideri g^{53} e si decomponga l'esponente in binario

$$53 = (110101)_2 = 32 + 16 + 4 + 1$$

A questo punto possiamo calcolare

$$g^{53} = g^{32} * g^{16} * g^4 * g^1$$

Sfruttando $g^{2x} = g^x * g^x$, calcoliamo rapidamente g^{32} come

$g \rightarrow g^2 \rightarrow g^4 \rightarrow g^8 \rightarrow g^{16} \rightarrow g^{32}$ (5 moltiplicazioni)

E quindi per calcolare $g^{53} = g^{32} * g^{16} * g^4 * g^1$ servono solo altre 3 moltiplicazioni visto che conosco g^{16}, g^4, g^1

Quindi g^{53} si calcola con 8 moltiplicazioni invece che con 52

Aumento e riduzione dei bit in bin

Aumento dei bit

- premettendo in modo progressivo un bit 0 a sinistra, il valore del numero non muta
- $(4)_{10} = (100)_2 = (0100)_2 = (00100)_2 = (000000000100)_2$
- $(5)_{10} = (101)_2 = (0101)_2 = (00101)_2 = (000000000101)_2$

Riduzione dei bit

- cancellando in modo progressivo un bit 0 a sinistra, il valore del numero non muta, ma bisogna arrestarsi quando si trova un bit 1!
- $(7)_{10} = (00111)_2 = (0111)_2 = (111)_2$ STOP !
- $(2)_{10} = (0010)_2 = (010)_2 = (10)_2$ STOP !

Somma tra Intero Positivi

Somma in base 2

Somma tra Numeri Binari

Si eseguono «in colonna» e si opera cifra per cifra

Si considera il riporto come per i decimali

- 0 + 0 = 0 riporto 0
- 1 + 0 = 1 riporto 0
- 0 + 1 = 1 riporto 0
- 1 + 1 = 0 riporto 1

Occorre sommare il riporto della cifra precedente

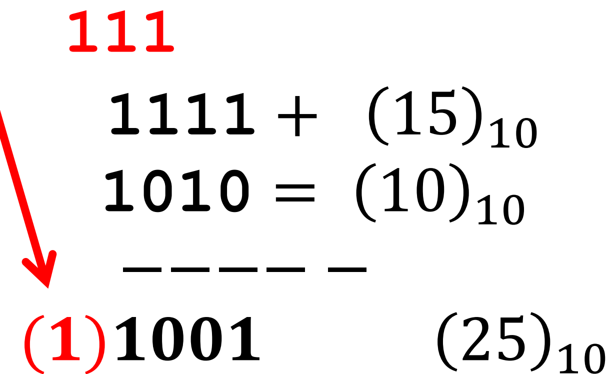
$$\begin{array}{ll} \ \ \color{red}{\mathbf{1}} & \\ \mathbf{0101}\ + & (5)_{10} \\ \mathbf{1001}\ = & (9)_{10} \\ ----\ - & \\ \mathbf{1110} & (14)_{10} \end{array} \qquad \begin{array}{ll} \color{red}{\mathbf{111}} & \\ \mathbf{1111}\ + & (15)_{10} \\ \mathbf{1010}\ = & (10)_{10} \\ ----\ - & \\ \color{red}{(1)}\mathbf{1001} & (25)_{10} \end{array}$$

Somma tra Numeri Binari

A volte i bit utilizzati per codificare gli addendi non bastano a contenere il risultato

- In questi casi occorrono più bit per codificare il risultato
- Si ha quindi un bit di **carry** (riportato nello State Register della ALU)

$$
\begin{array}{lll}
 & {\color{red}\mathbf{1}} & \\
\mathbf{0101} & + & (5)_{10} \\
\mathbf{1001} & = & (9)_{10} \\
\hline
\mathbf{1110} & & (14)_{10}
\end{array}
$$



I numeri Interi: Modulo e Segno

Positivi e Negativi

Rappresentazione Modulo e Segno

È possibile dedicare **il primo bit** alla codifica del **segno**

- **"1"** il numero che segue è negativo
- **"0"** il numero che segue è positivo

Con m cifre in binario e codifica modulo dedico 2^{m-1} per i positivi e 2^{m-1} per gli stessi cambiati di segno

- posso rappresentare tutti i numeri nell'intervallo

$$X \in \left[-2^{m-1} + 1, 2^{m-1} - 1\right]$$

Es

- **01010 = + 10**
- **11101 = - 13**
- **-27 = 111011**
- **122 = 01111010**

Rappresentazione Modulo e Segno

Esempio $m = 3$

- 0 = 000
- 1 = 001
- 2 = 010
- 3 = 011
- -0 = 100
- -1 = 101
- -2 = 110
- -3 = 111

Ho due codifiche differenti lo zero

C'è uno «spreco» nella codifica

Ostacola realizzazione circuitale delle operazioni algebriche (non lo mostriamo)

Occorre trovare una rappresentazione migliore!

I numeri Interi: Complemento a Due

Positivi e Negativi

Rappresentazione in Complemento a 2 (CP2)

Date m cifre binarie, disponibili 2^m configurazioni distinte

In CP2 se ne usano:

- $2^{m-1} - 1$ per valori positivi
- 1 per lo zero
- 2^{m-1} per i valori negativi

Con m bit rappresento l'intervallo $[-2^{m-1}, 2^{m-1} - 1]$

Rappresentazione in Complemento a 2 (CP2)

Sia $X \in [-2^{m-1}, 2^{m-1} - 1]$ il numero da rappresentare in CP2, con m bit.

- se X è **positivo** o nullo **scrivo** $\boldsymbol{X}$ **in binario con** $\boldsymbol{m}$ **bit**
- se X è **negativo scrivo** $\mathbf{2}^{\boldsymbol{m}} - |\boldsymbol{X}|$ in binario con $\boldsymbol{m}$ **bit**

Questo equivale alla seguente codifica:

$$N_{CP2} = a_{m-1}a_{m-2} \dots a_1 a_0$$

$$= -a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

$$= -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i \ , \qquad a_i \in \{0,1\}$$

Rappresentazione in Complemento a 2 (CP2)

Sia $X \in [-2^{m-1}, 2^{m-1}-1]$ il numero da rappresentare in CP2, con m bit.

- se X è **positivo** o nullo **scrivo** $\boldsymbol{X}$ in binario con $\boldsymbol{m}$ **bit**
- se X è **negativo scrivo** $\mathbf{2^m} - |\boldsymbol{X}|$ in binario con $\boldsymbol{m}$ **bit**

Questo equivale alla seguente codifica:

$$N_{CP2} = a_{m-1}a_{m-2} \dots a_1 a_0$$

$$= \boxed{-a_{m-1} \times 2^{m-1}} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$

$$= \boxed{-a_{m-1} \times 2^{m-1}} + \sum_{i=0}^{m-2} a_i \times 2^i \,, \qquad a_i \in \{0,1\}$$

i.e., viene cambiato il segno dell'addendo relativo alla cifra più significativa

Rappresentazione in CP2

Esempio $m = 3$

- `-4 = 100`
- `-3 = 101`
- `-2 = 110`
- `-1 = 111`
- ` 0 = 000`
- ` 1 = 001`
- ` 2 = 010`
- ` 3 = 011`

Rappresentazione in CP2

Con i positivi copro solo il range $[0, 2^{m-1}-1]$, quindi la prima cifra è 0 (il numero è minore di 2^{m-1})

Con i negativi copro il range $[-2^{m-1}, -1]$ e scrivo $2^m - |X|$, e quindi la prima cifra è 1 (il numero è maggiore di 2^{m-1})

Quindi, il primo bit **indica il segno** del numero

- Attenzione: questo numero **non è il segno**: **cambiandolo non** si ottiene il **numero opposto**
- $45 = (0101101)_{CP2}$ se cambio di segno alla prima cifra
- $(1101101)_{CP2} \rightarrow -2^6 + 2^5 + 2^3 + 2^2 + 1 =$
  $= -64 + 45 = -19$

Inoltre, un solo valore per lo 0 (cioè m volte 0), nessuna configurazione "sprecata" dalla codifica

Rappresentazione in CP2

Es, definire un intervallo che contenga -23 e 45

Rappresentazione in CP2

Es, definire un intervallo che contenga -23 e 45

- $m = 7$, copro $[-2^6, 2^6 - 1] = [-64, 63]$
- ~~$m = 6$, copro $[-2^5, 2^5 - 1] = [-32, 32]$~~ (non cont. 45)
- $-23 \rightarrow 2^7 - 23 = 128 - 23 = 105 = (1101001)_{CP2}$
- $45 = (0101101)_{CP2}$

NB: occorre utilizzare **sempre $\boldsymbol{m}$ bit**. Se non avessi messo lo 0 iniziale in $(45)_{CP2}$ avrei ottenuto un numero negativo a 6 bit!

Conversione Decimale ➡ CP2

Metodo "operativo" per rappresentare X ad m bit

1. Controllo che $X \in [-2^{m-1}, 2^{m-1} - 1]$, altrimenti m bit non bastano
2. Se X è positivo, scrivo X utilizzando m bit
   NB: ricordandosi di aggiungerei zeri se necessario all'inizio del numero!
3. Se X è negativo:
   a) Scrivo $|X|$ utilizzando m bit
   b) **Complemento** tutti i bit di X $(1 \rightarrow\ 0, 0 \rightarrow\ 1)$
   c) **Sommo 1** al numero ottenuto

Cambio di segno un numero in C2

L'inverso additivo (o opposto) $-N$ di un numero N rappresentato in C2 si ottiene:

- Complementando (negando) ogni bit del numero
- Sommando 1 alla posizione meno significativa

Esempio:

- $(01011)_{CP2} = 1\times2^3+1\times2^1+1\times2^0 = 8+2+1 = (11)_{10}$
- $10100 + 1 = 10101_{CP2} = -1\times2^4+1\times2^2+1\times2^0 =$
  $-16+4+1 = -11_{dec}$

Si provi a invertire $11011_{CP2} = -5_{dec}$

Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari

Esempi Conversione Decimale ➡ CP2

Esempio: scrivere -56 in CP2 con il numero di bit necessari

$m = 7$ copre $[-2^6, 2^6 - 1] = [-64\,, 63]$

Scrivo $(56)_{10} \rightarrow 0111000$

Complemento $\rightarrow 1000111$

Sommo 1 $\quad 1$

$(1001000)_{CP2}$ = $(-56)_{10}$

56	0
28	0
14	0
7	1
3	1
1	1
0	

Esercizi

Esercizio: convertire in complemento a 2 i seguenti numeri, utilizzando il numero di bit necessario per esprimerli tutti

$$(12)_{10} =$$
$$(-12)_{10} =$$
$$(-8)_{10} =$$
$$(1)_{10} =$$
$$(-101)_{10} =$$
$$(-54)_{10} =$$

Esercizi

Esercizio: convertire in complemento a 2 i seguenti numeri, utilizzando il numero di bit necessario per esprimerli tutti

$$(12)_{10} = (0000\,1100)_{CP2}$$
$$(-12)_{10} = (1111\,0100)_{CP2}$$
$$(-8)_{10} = (1111\,1000)_{CP2}$$
$$(1)_{10} = (0000\,0001)_{CP2}$$
$$(-101)_{10} = (1001\,1011)_{CP2}$$
$$(-54)_{10} = (1100\,1010)_{CP2}$$

Conversione CP2 ➡ Decimale

Possiamo utilizzare la definizione

$$N_{CP2} = a_{m-1}a_{m-2} \dots a_1 a_0$$
$$= -a_{m-1} \times 2^{m-1} + a_{m-2} \times 2^{m-2} + \dots + a_0 \times 2^0$$
$$= -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i \ , \qquad a_i \in \{0,1\}$$

Es $(1001000)_{CP2} = -2^6 + 2^3 = -64 + 8 = (-56)_{10}$

$$(10011011)_{CP2} = -2^7 + 2^4 + 2^3 + 2^1 + 2^0 =$$
$$= -128 + 16 + 8 + 2 + 1 = (-101)_{10}$$

NB convertite sempre in decimale con questo metodo per controllare le vostre operazioni

Conversione CP2 ➡ Decimale

… in alternativa è possibile utilizzare un metodo operativo:

1. Se $(X)_{CP2}$ inizia per 0, allora è positivo: lo converto normalmente
2. Se $(X)_{CP2}$ inizia per 1, allora è negativo
   a) **Complemento** tutti i bit di $(X)_{CP2}$ $(1 \rightarrow\ 0, 0 \rightarrow\ 1)$
   b) **Sommo 1** al numero ottenuto
   c) **Converto** in decimale e cambio di segno

Esempio

Esercizio: riconvertire in decimale i seguenti numeri in complemento a 2

$(1001010)_{CP2} \rightarrow (0110101) \rightarrow (0110110) \rightarrow (-54)_{10}$

$(1001010)_{CP2} = -2^6 + 2^3 + 2^1 = -64 + 8 + 2 = -54$

$(011)_{CP2}$

$(1101001)_{CP2}$

$(11111)_{CP2}$

$(10100)_{CP2}$

$(101)_{CP2}$

Aumento e riduzione dei bit in C_2

Estensione del segno:

- *replicando* in modo progressivo il primo bit, il valore del numero non muta

$4 = 0100 = 00100 = 00000100 = \dots$ (indefinitamente)

$-5 = 1011 = 11011 = 11111011 = \dots$ (indefinitamente)

$-1\times2^3+1\times2^1+1\times2^0$

$-1\times2^4+1\times2^3+1\times2^1+1\times2^0$

Contrazione del segno:

- *cancellando* in modo progressivo il primo bit, il valore del numero non muta, *purché il bit di segno non abbia a invertirsi* !

$7 = 000111 = 00111 = 0111$ **STOP!** (111 è < 0)

$-3 = 111101 = 11101 = 1101 = 101$ **STOP!** (01 è > 0)

Somma Tra Interi

Riconoscere Overflow

Somma tra Numeri in CP2

In CP2 l'operazione di somma si realizza **come nella rappresentazione binaria posizionale**

Grazie alla rappresentazione in CP2 **è possibile eseguire** anche **sottrazioni** tra numeri binari con lo stesso meccanismo (i.e., somme tra interi di segno opposto)

Esempio no overflow

Esempio: 60 − 54

diventa 60 + (−54)

$$
\begin{array}{rcl}
 & & 1\;1\;\;1\;1 \\
(60)_{10} & = & (0\;1\;1\;1\;1\;0\;0)_{CP2} \\
(-54)_{10} & = & (1\;0\;0\;1\;0\;1\;0)_{CP2} \\
\hline
 & & (1)\;0\;0\;0\;0\;1\;1\;0
\end{array}
$$

Il riporto (carry) viene ignorato

Quando sommo numeri di segno opposto non può esserci overflow

Il risultato è positivo $(0000110)_{CP2} = (6)_{10}$

Esempio

Esempio: $(100)_{CP2} + (101)_{CP2}$

1

1 0 0 +

1 0 1 =

[1] (1) 0 0 1

Ignoro il bit di carry

Overflow: la somma di due numeri negativi mi ha dato un numero positivo.

L'overflow si indica quadre: [1] c'è overflow, [0] non c'è

Il risultato non ha senso, occorre scrivere gli addendi con un bit in più per rappresentare il risultato dell'operazione

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

-3 =>
-4 =>
-7 =>

-3 =>
+6 =>
+3 =>

-3 =>
-7 =>
-10 =>

+2 =>
+5 =>
+7 =>

+3 =>
+6 =>
+9 =>

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

```
-3  =>        1101
-4  =>        1100
------------------
-7  => [0] (1)1001

-3  =>
+6  =>
------
+3  =>
```

```
-3  =>
-7  =>
------
-10 =>

+2  =>
+5  =>
------
+7  =>

+3  =>
+6  =>
------
+9  =>
```

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

```
-3  =>         1101
-4  =>         1100
-------------------
-7  =>  [0] (1)1001

-3  =>          1101
+6  =>          0110
--------------------
+3  =>  [0]  (1)0011
```

```
-3  =>
-7  =>
---------
-10 =>

+2  =>
+5  =>
---------
+7  =>

+3  =>
+6  =>
---------
+9  =>
```

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

```
-3  =>          1101
-4  =>          1100
--------------------
-7  =>  [0] (1)1001

-3  =>          1101
+6  =>          0110
--------------------
+3  =>  [0] (1)0011

-3  =>          1101
-7  =>          1001
--------------------
-10 =>  [1](1)  0110

+2  =>
+5  =>
--------
+7  =>

+3  =>
+6  =>
--------
+9  =>
```

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

```
-3  =>          1101
-4  =>          1100
--------------------
-7  =>  [0] (1)1001

-3  =>          1101
+6  =>          0110
--------------------
+3  =>  [0]  (1)0011

-3  =>           1101
-7  =>           1001
---------------------
-10 =>  [1](1)   0110

+2  =>           0010
+5  =>           0101
---------------------
+7  =>  [0](0)   0111

+3  =>
+6  =>
---------
+9  =>
```

Esempi

Esempi: con $m = 4$ bit

Indico tra () bit di carry, tra [] bit di overflow

```
-3   =>         1101
-4   =>         1100
---------------------
-7   =>  [0] (1)1001

-3   =>          1101
+6   =>          0110
---------------------
+3   =>  [0]  (1)0011

-3   =>           1101
-7   =>           1001
----------------------
-10  =>  [1](1)   0110

+2   =>           0010
+5   =>           0101
----------------------
+7   =>  [0](0)   0111

+3   =>           0011
+6   =>           0110
----------------------
+9   =>  [1](0)   1001
```

Carry e Overflow in CP2

In CP2 occorre **ignorare il bit di carry**, cioè il riporto che cade sul bit di segno

In CP2 occorre individuare **l'overflow**, i.e., casi in cui il risultato è fuori dall'intervallo rappresentabile con i bit utilizzati.

Quando c'è **overflow il risultato è inconsistente** con gli addendi:

- Somma di due addendi positivi da un numero negativo
- Somma di due addendi negativi da un numero positivo

NB non può esserci overflow quando sommo due numeri di segno opposto

Carry e Overflow in CP2 (addizione algebrica)

Si ha overflow quando il risultato corretto dell'addizione eccede il potere di rappresentazione dei bit a disposizione

Si può avere overflow "senza carry"

- Capita quando da due addendi positivi otteniamo un risultato negativo, es:

+3 => 0011

+6 => 0110

+9 => [1](0) 1001

Si può avere un "carry" senza overflow

- Può essere un innocuo effetto collaterale
- Capita quando due addendi discordi generano un risultato positivo, es:

-3 => 1101

+6 => 0110

+3 => [0] (1)0011

Interi relativi in Modulo e Segno ed in CP2

Se usiamo 1 Byte: da –128 a 127

dec.	m&s	C_2
127	**0**1111111	**0**1111111
126	**0**1111110	**0**1111110
…	…	…
2	**0**0000010	**0**0000010
1	**0**0000001	**0**0000001
+0	**0**0000000	**0**0000000
-0	**1**0000000	**-**
-1	**1**0000001	**1**1111111
-2	**1**0000010	**1**1111110
…	…	…
-126	**1**1111110	**1**0000010
-127	**1**1111111	**1**0000001
-128	**-**	**1**0000000

Proprietà del CP2

Si verifichi che

i) con due applicazioni dell'algoritmo di cambio del segno si ottiene il numero iniziale, i.e.,[$-(-N) = N$]

ii) che lo zero in C2 è (correttamente) opposto di se stesso [$-0 = 0$]

Esercizi su CP2

Positivi e Negativi

TDE 11/2011

1. Si fornisca la codifica binaria CP_2 del numero -221 utilizzando il minor numero di bit necessari per una corretta rappresentazione

Punto 1, rappresentazione in CP_2 di -221

221 richiede 9 bit perchè così copro $[-2^8, 2^8 - 1]$ con 8 bit copro solamente $[-2^7, 2^7 - 1]$. Quindi codifico 221

221	1
110	0
55	1
27	1
13	1
6	0
3	1
1	1
0	

Punto 1, rappresentazione in CP_2 di -221

221 richiede 9 bit perchè così copro $[-2^8, 2^8 - 1]$ con 8 bit copro solamente $[-2^7, 2^7 - 1]$. Quindi codifico 221

```
221  1
110  0
55   1
27   1
13   1
6    0
3    1
1    1
0
```

```
221 = 11011101
     011011101   (lo metto a 9 bit)
     100100010   (complemento)
     100100011   (sommo 1)
```

Esempio

a) Si dica qual è l'intervallo di valori interi rappresentabile con la codifica in complemento a due a 9 bit.

b) Con riferimento a tale codifica indicare, giustificando brevemente le risposte, quali delle seguenti operazioni possono essere effettuate correttamente:

   i. -254 – 255

   ii. + 254 – 253

   iii. -18 + 236

   iv. + 217 + 182

c) Mostrare in dettaglio come avviene il calcolo delle operazioni (i) e (ii), evidenziando il bit di riporto e il bit di overflow così ottenuti. (Il bit di overflow è pari ad 1 se si verifica overflow, o altrimenti.)

Esempio

a. Valori rappresentabili vanno da -256 a +255.
b. Le soluzioni:
   i. -254 – 255: NO, si ottiene un valore negativo troppo grande in valore assoluto
   ii. + 254 – 253: SI, si ottiene un valore piccolo in valore assoluto
   iii. -18 + 236: SI, si ottiene un valore positivo, grande in valore assoluto ma nei limiti
   iv. + 217 + 182: NO, si ottiene un valore positivo troppo grande in valore assoluto
c.

```
      100000010  (-254)
      100000001  (-255)
      -----------------
[1](1)000000011  (-509)
```

```
      011111110  (+254)
      100000011  (-253)
      -----------------
[0](1)000000001  (+1)
```

I Numeri Frazionari

I Numeri Frazionari

Un'approssimazione dei numeri reali in [0,1]

Si rappresentano anteponendo **0.** al numero

$$N_p = (0.a_1 a_2 \dots a_m)_p =$$

$$N_p = a_1 \times p^{-1} + a_2 \times p^{-2} \dots + a_m \times p^{-m}$$

$$N_p = \sum_{i=1}^{m} a_i \times p^{-i} , \qquad a_i \in A_p$$

In base 10

$$0.586 = 5 \times 10^{-1} + 8 \times 10^{-2} + 6 \times 10^{-3}$$

In base 2

$$(0.101)_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = \frac{1}{2} + \frac{1}{8} = (0.625)_{10}$$

I Numeri Frazionari : Osservazioni

Date m cifre in base $p = 2$, posso rappresentare **un sottoinsieme** dell'intervallo continuo $[0\,,1\;-2^{-m}]$.

L'errore di approssimazione sarà minore di 2^{-m}

Conversione Decimale ➡ Binario su Frazionari

L'algoritmo delle **moltiplicazioni successive** per convertire $N_{10} \in [0,1]$

1. Moltiplico N_{10} per 2.
2. La parte intera del risultato definisce una cifra della rappresentazione binaria finale
3. la parte frazionaria risultante viene moltiplicata per 2
4. Si itera 1- 3 fino a
   - Ottenere parte frazionaria nulla (rappresentazione esatta) **oppure**
   - Coprire tutte le cifre binarie a disposizione (rappresentazione approssimata)
5. **La cifra più significativa** (il coefficiente di 2^{-1}) è dato dalla **prima parte intera calcolata, quella meno significativa** (il coefficiente di 2^{-m}) è dato **dall'ultima parte intera calcolata**

Esempio

Convertire in binario 0.625 utilizzando $m = 6$ bit

$0.625 \times 2 = 1 + 0.25$

$0.250 \times 2 = 0 + 0.5$

$0.500 \times 2 = 1 + 0$

0

Parte intera + parte frazionaria

La parte intera definisce la rappresentazione binaria

Otteniamo $(0.625)_{10} = 0.101$, la rappresentazione è esatta

Se dovessi usare 6 bit $(0.625)_{10} = 0.101000$

Esempio

Convertire in binario 0.587 utilizzando $m = 6$ bit

$$0.587 \times 2 = 1 + 0.174$$
$$0.174 \times 2 = 0 + 0.348$$
$$0.348 \times 2 = 0 + 0.696$$
$$0.696 \times 2 = 1 + 0.392$$
$$0.392 \times 2 = 0 + 0.784$$
$$0.784 \times 2 = 1 + 0.560$$

Parte intera + parte frazionaria

La parte intera definisce la rappresentazione binaria

Otteniamo $(0.587)_{10} \approx 0.100101$

Rappresentazione approssimata, la parte frazionaria finale non è 0. L'accuratezza è di almeno 2^{-6}.

Esempio

Convertire in binario 0.9 utilizzando $m = 16$ bit

- $0.9 \times 2 = 1 + 0.8$
- $0.8 \times 2 = 1 + 0.6$
- $0.6 \times 2 = 1 + 0.2$
- $0.2 \times 2 = 0 + 0.4$
- $0.4 \times 2 = 0 + 0.8$
- $0.8 \times 2 = 1 + 0.6$
- $0.6 \times 2 = 1 + 0.2$
- $0.2 \times 2 = 0 + 0.4$
- $0.4 \times 2 = 0 + 0.8$
- $0.8 \times 2 = 1 + 0.6$
- ….

Rappresentazione **periodica**! Inutile procedere oltre

Otteniamo $(0.9)_{10} \approx 0.1\ 1100\ 1100\ 1100\ 110$ con accuratezza di almeno 2^{-16}.

Rappresentazione in Virgola Fissa

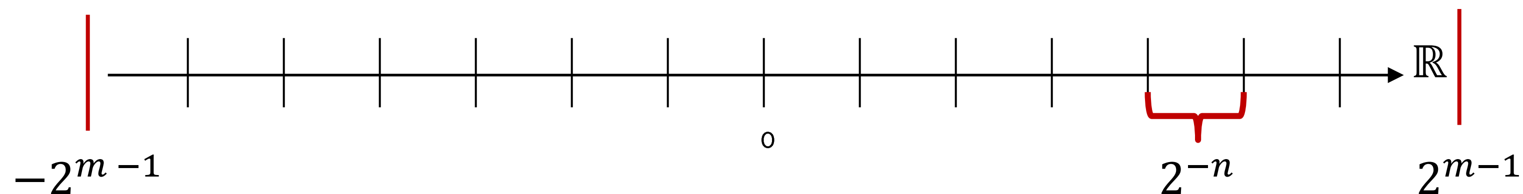
Si definiscono m bit per la parte intera e n bit per la parte frazionaria e si scrivono le due parti indipendentemente

Rappresentare $(-123{,}21)_{10}$ utilizzando $m = 8, n = 6$ e rappresentazione in CP_2 per la parte intera

$$(-123)_{10} = (10000101)_{CP2}$$
$$(0{,}21)_{10} \approx (001101)_2$$
$$-123{,}21_{10} \approx (10000101.001101)_2$$

Questa rappresentazione mi da

- Precisione costante lungo l'asse reale $\mathbb{R}$:
- Estremi definiti solo da m



Numeri frazionari in virgola fissa

La sequenza di bit rappresentante un numero frazionario consta di **due parti di lunghezza prefissata**

- Il numero di bit a sinistra e a destra della virgola è stabilito a priori, anche se alcuni bit restassero nulli

È un sistema di **rappresentazione semplice**, ma poco flessibile. **Comporta sprechi di bit**

- Per rappresentare in virgola fissa numeri molto grandi (o molto precisi) occorrono molti bit
- **La precisione nell'intorno dell'origine e lontano dall'origine è la stessa**
  - Anche se su numeri molto grandi in valore assoluto la parte frazionaria può non essere particolarmente significativa

Virgola mobile (floating point)

Il numero r in base p in virgola mobile è espresso come:

$$r \ = \pm M \cdot b^n$$

- M mantissa (parte frazionaria, è un numero razionale)
- b: base della notazione esponenziale (numero naturale)
- n: caratteristica (numero intero)
- $\boldsymbol{M}$ **e** $\boldsymbol{n}$ **sono in base** $\boldsymbol{p}$ (non necessariamente 10)

Esempio $(p = 10, b = 10)$:

$-331{,}6875 = -0{,}3316875 \cdot 10^3 \quad M = \ -0{,}3316875; \ \ n \ = \ 3$

Il numero può essere codificato usando **un numero predefinito di bit** per M e per n

- b e p non devono essere rappresentati, sono definiti dallo standard

Numeri frazionari in virgola mobile

Esempio

- Supponiamo: $b = 2,\ 3$ bit per M, 3 bit per n, M ed n in binario naturale

  $M = (011)_2$ ed $n = (010)_2$

  $$R_{virgola\ mobile} = 0{,}011 \times 2^{010} = (1/4 + 1/8) \times 2^2 = 3/8 \times 4 = 3/2 = 1{,}5_{dec}$$

M ed n possono anche essere negativi

- Normalmente infatti si usa il *modulo e segno* per M, mentre per n si usa la rappresentazione cosiddetta *in eccesso* (qui non spiegata)

Vantaggi della virgola mobile

- si possono rappresentare con pochi bit numeri molto grandi **oppure** molto precisi (cioè con molti decimali)
- Sull'asse dei valori i numeri rappresentabili si affollano nell'intorno dello zero, e sono sempre più sparsi al crescere del valore assoluto

Proprietà fondamentale

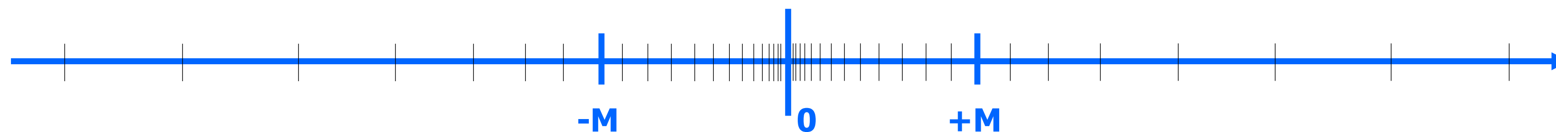
I circa 4 miliardi di configurazioni dei 32 bit usati consentono di coprire un campo di valori molto ampio grazie alla distribuzione non uniforme.

Per numeri piccoli in valore assoluto valori rappresentati sono «fitti»,

Per numeri grandi in valore assoluto valori rappresentati sono «diradati»

Approssimativamente gli intervalli tra valori contigui sono

- per valori di 10000 l'intervallo è di un millesimo
- per valori di 10 milioni l'intervallo è di un'unità
- per valori di 10 miliardi l'intervallo è di mille



Standard IEEE 754-1985 per i numeri float

Quattro diversi formati, differiscono nel numero totale dei bit utilizzati. Quelli più diffusi:

- precisione singola: 32 bit
- precisione doppia: 64 bit
- precisione tripla: 128 bit

Si usa sempre base $b = 2$ e $p = 2$

Codifica di Testi ed Immagini

Rappresentazione dei Caratteri

Ogni carattere viene mappato in un numero intero (che è espresso da sequenza di bit) utilizzando dei codici

Il codice più usato è l'*ASCII (American Standard Code for Information Interchange)* a 7 o 8 bit che contiene:

- Caratteri alfanumerici
- Caratteri simbolici (es. punteggiatura, **@&%$** etc..)
- Caratteri di comando (es. termina riga, vai a capo, tab)

Non solo numeri codifica dei caratteri

Nei calcolatori i caratteri vengono codificati mediante sequenze di $n \geq 1$ bit, ognuna rappresentante un carattere distinto

- Corrispondenza biunivoca tra numeri e caratteri

Codice ASCII (American Standard Computer Interchange Interface): utilizza n=7 bit per 128 caratteri

Il codice ASCII a 7 bit è pensato per la lingua inglese.

Codifica ASCII esteso a 8 bit (256 parole di codice). È la più usata. Rappresenta il doppio dei caratteri

- Si aggiungono così, ad esempio, le lettere con i vari gradi di accento (come À, Á, Â, Ã, Ä, Å, ecc..), necessarie in molte lingue europee, e altri simboli speciali ancora

La codifica ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z

Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)
16	10	020	DLE	(data link escape)
17	11	021	DC1	(device control 1)
18	12	022	DC2	(device control 2)
19	13	023	DC3	(device control 3)
20	14	024	DC4	(device control 4)
21	15	025	NAK	(negative acknowledge)
22	16	026	SYN	(synchronous idle)
23	17	027	ETB	(end of trans. block)
24	18	030	CAN	(cancel)
25	19	031	EM	(end of medium)
26	1A	032	SUB	(substitute)
27	1B	033	ESC	(escape)
28	1C	034	FS	(file separator)
29	1D	035	GS	(group separator)
30	1E	036	RS	(record separator)
31	1F	037	US	(unit separator)

Dec	Hx	Oct	Html	Chr
32	20	040		Space
33	21	041	!	!
34	22	042	"	"
35	23	043	#	#
36	24	044	$	$
37	25	045	%	%
38	26	046	&	&
39	27	047	'	'
40	28	050	(	(
41	29	051	)	)
42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<
61	3D	075	=	=
62	3E	076	>	>
63	3F	077	?	?

Dec	Hx	Oct	Html	Chr
64	40	100	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z
91	5B	133	[	[
92	5C	134	\	\
93	5D	135	]	]
94	5E	136	^	^
95	5F	137	_	_

Dec	Hx	Oct	Html	Chr
96	60	140	`	`
97	61	141	a	a
98	62	142	b	b
99	63	143	c	c
100	64	144	d	d
101	65	145	e	e
102	66	146	f	f
103	67	147	g	g
104	68	150	h	h
105	69	151	i	i
106	6A	152	j	j
107	6B	153	k	k
108	6C	154	l	l
109	6D	155	m	m
110	6E	156	n	n
111	6F	157	o	o
112	70	160	p	p
113	71	161	q	q
114	72	162	r	r
115	73	163	s	s
116	74	164	t	t
117	75	165	u	u
118	76	166	v	v
119	77	167	w	w
120	78	170	x	x
121	79	171	y	y
122	7A	172	z	z
123	7B	173	{	{
124	7C	174			|
125	7D	175	}	}
126	7E	176	~	~
127	7F	177		DEL

www.LookupTables.com

Source: www.LookupTables.com

Rilevare gli errori

Spesso, al codice ASCII a 7 bit è associato un ottavo bit come **bit di parità.** I calcolatori infatti hanno parole di memoria da un Byte o suoi multipli.

Il bit di parità serve per rilevare eventuali errori che potrebbero avere alterato la sequenza di bit, purché siano errori di tipo abbastanza semplice

Bit di parità

Si aggiunge un bit extra, in modo che il numero di bit uguali a 1 sia sempre pari:

- 1100101 (quattro bit 1) $\Rightarrow$ 11001010 (quattro bit 1)
- 0110111 (cinque bit 1) $\Rightarrow$ 01101111 (sei bit 1)

Se per errore un (solo) bit si inverte, il conteggio dei bit uguali a 1 dà valore dispari!

Così si può rilevare l'esistenza di un errore da un bit (ma non localizzarne la posizione)

Aggiungendo più bit extra (secondo schemi opportuni) si può anche localizzare l'errore.

Il bit di parità non rileva gli errori da due bit; ma questi ultimi sono meno frequenti di quelli da un bit

Convertire il messaggio pubblicitario in base 10 assumendo sia scritto in CP2

MediaOne

PER TUTTI I CLIENTI
ENEL ENERGIA LUCE
OFFERTA GAS
-20%
PER 12 MESI HAI
IL 20% DI SCONTO
SULLA COMPONENTE MATERIA PRIMA GAS
CHIAMA ENEL ENERGIA
800 900 860

PER TUTTI GLI ALTRI
UN CODICE BINARIO

01100101 01101110
01100101 01110010
01100111 01101001
01100001 00001101

enel.it

enel

Photo Credits: Andrea Sanfilippo

Immagini RGB

$I \in \mathbb{R}^{R \times C \times 3}$

$R \in \mathbb{R}^{R \times C}$

$G \in \mathbb{R}^{R \times C}$

$B \in \mathbb{R}^{R \times C}$

Immagini RGB

Le immagini sono salvate codificando i valori di ogni piano del colore (R,G,B) in 8 bits, quindi in valori [0,255]

$R \in \mathbb{R}^{R \times C}$

123	122	134	121	132
122	121	125	132	124
119	127	137	119	139

Immagini RGB

Codifica delle Immagini

Esistono diverse codifiche dell'immagine, non sempre questa viene scritta pixel per pixel.

È spesso conveniente rappresentare l'immagine secondo codifiche che permettano di ridurre le dimensioni.

Codifiche *lossless*: permettono, senza perdita di informazione, di comprimere l'immagine

- e.g, non serve ripetere il valore nelle aree costanti, conviene registrare le variazioni (e.g. gif, png)

Codifiche *lossy:* perdita di informazione e di qualità

- e.g., le immagini jpeg sono compresse a blocchi, ogni blocco contiene meno dettagli dell'immagine originale.