# Università degli Studi di Cassino e del Lazio Meridionale



## Corso di Calcolatori Elettronici

*Rappresentazione dei dati numerici*
*Aritmetica dei registri*

Anno Accademico 2013/2014

Alessandra Scotto di Freca

Si ringrazia il prof.Francesco Tortorella per il materiale didattico

# BIG IDEA: Bits can represent anything!!

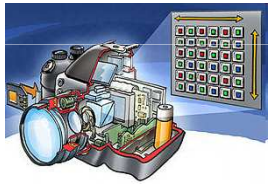
- Caratteri
  - 26 lettere $\Rightarrow$ 5 bits ($2^5 = 32$)
  - Minuscole/maiuscole + punteggiatura
    $\Rightarrow$ 7 bits (in 8) (“ASCII”)
  - Codice standard per rappresentare tutti I linguaggi del mondo
    $\Rightarrow$ 8,16,32 bits (“Unicode”)
    `www.unicode.com`
- Valori logici
  - 0 $\Rightarrow$ False, 1 $\Rightarrow$ True
- Colori
  - 3 valori di intensità per i tre colori fondamentali RGB (3 x 8 bit = 24 bit)
- Locazioni / indirizzi comandi
- **Ricorda**: N bits $\Rightarrow$ al più $2^N$ oggetti

# Rappresentazione analogica o digitale?

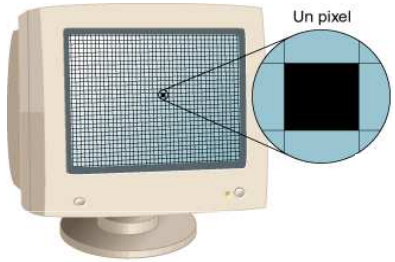
La rappresentazione analogica cerca di riprodurre alcune caratteristiche del fenomeno rappresentato segnalando ogni loro minima variazione:
*è una rappresentazione continua*

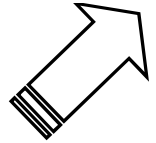
La rappresentazione digitale viene da digit numero è una rappresentazione numerica, discreta

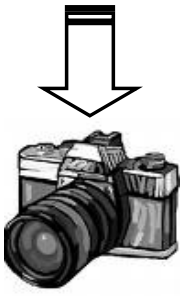
Le macchine lavorano con le sole informazioni digitali, perché per memorizzare le informazioni fanno uso di un dispositivo elettronico bistabile (dispositivo che può assumere uno tra due stati stabili associati a due livelli differenti di tensione).

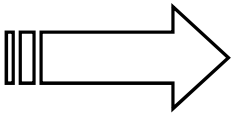


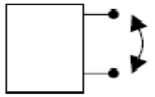
sensori (CCD, CMOS) sostituiscono la pellicola fotosensibile



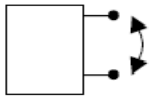









$V > V_0$ memorizzato 1



$V < V_0$ memorizzato 0

Le macchine fotografiche analogiche consentono solo lo sviluppo delle foto quelle digitali consentono alcune operazioni che una volta potevano essere svolte dai soli computer dopo la scansione e digitalizzazione

# Informazione multimediale

- I primi computer lavoravano con i numeri poi con le lettere permettendo editing di testo, già da decenni le informazioni sulle quali un elaboratore può lavorare sono immagini, video, suoni….
- L'occupazione di memoria e la trasmissione delle informazioni dipendono dalla loro diversa natura.

  quanti bit servono per rappresentare un carattere?

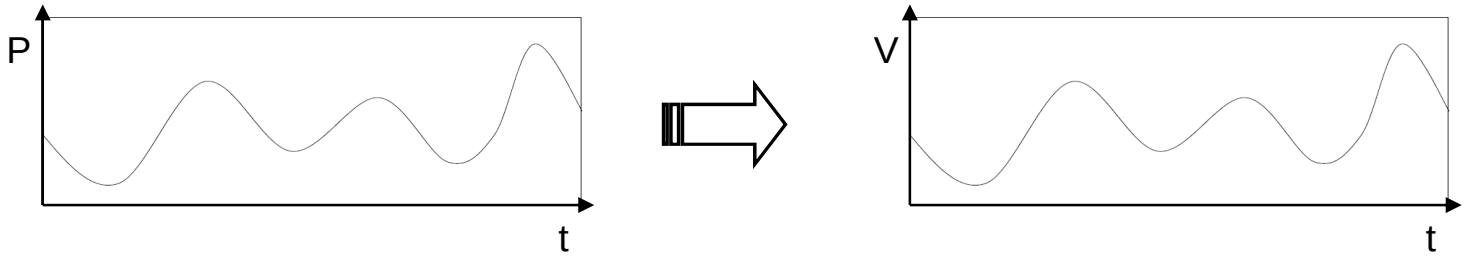
  quanti un immagine?

  quanto tempo impiegate a scaricare un video?

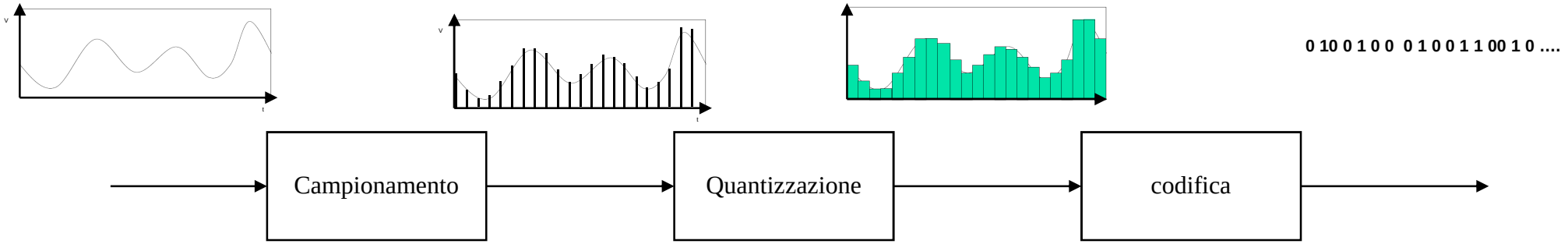
- Per rispondere a queste domande occorre andare studiare il processo di digitalizzazione: ossia quel processo che permette ad un elaboratore o ad un dispositivo multimediale di trasformare le informazioni in dati sui quali può agire.

# Dal continuo al discreto

- Consideriamo un segnale monodimensionale, ad esempio un suono.
  Quando sentiamo un suono il nostro timpano è perturbato da un onda di pressione
  - Un sistema di riproduzione analogica converte l'onda di pressione in un onda di tensione elettrica



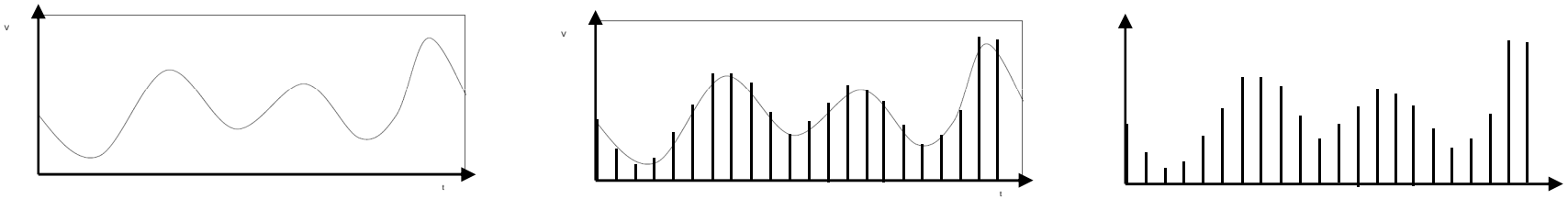

  - Un sistema digitale rappresenta un qualunque oggetto o fenomeno all'interno della memoria del computer tramite digit binari. Per effettuare questa trasformazione si effettua una conversione analogico-digitale che trasforma un segnale analogico (valori continui in un tempo continuo) in segnale numerico (valori discreti in tempo discreto).
    Questa operazione è costituita da tre fasi:



# Campionamento

•Campionare un segnale significa prelevare da esso dei “campioni” a frequenza regolare. L’ampiezza di un segnale (P o V) viene rappresentata da insieme numerico di cardinalità ridotta rispetto al segnale continuo originale.

•L’accuratezza del campionamento dipende dalla frequenza di campionamento che è il numero di campioni rilevato nell’unita tempo (*n* x 1/sec = *n* Hertz)



•Tanto più è alta la frequenza di campionamento tanto più precisa è la digitalizzazione dell’onda e tanto più spazio è richiesto per la memorizzazione.

•La frequenza di campionamento va scelta in modo tale da non permettere all’orecchio umano di percepire la differenza con l’originale.

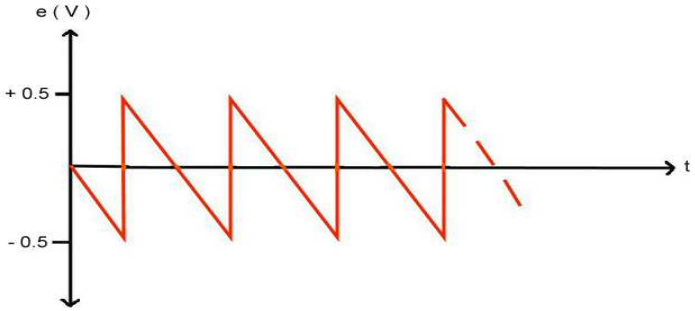
$fc_{\text{Telefono}}$= 8kHz, $fc\__{\text{riproduzioneCDaudio}}$=88kHz, $fc\__{\text{DVDaudio}}$=96kHz

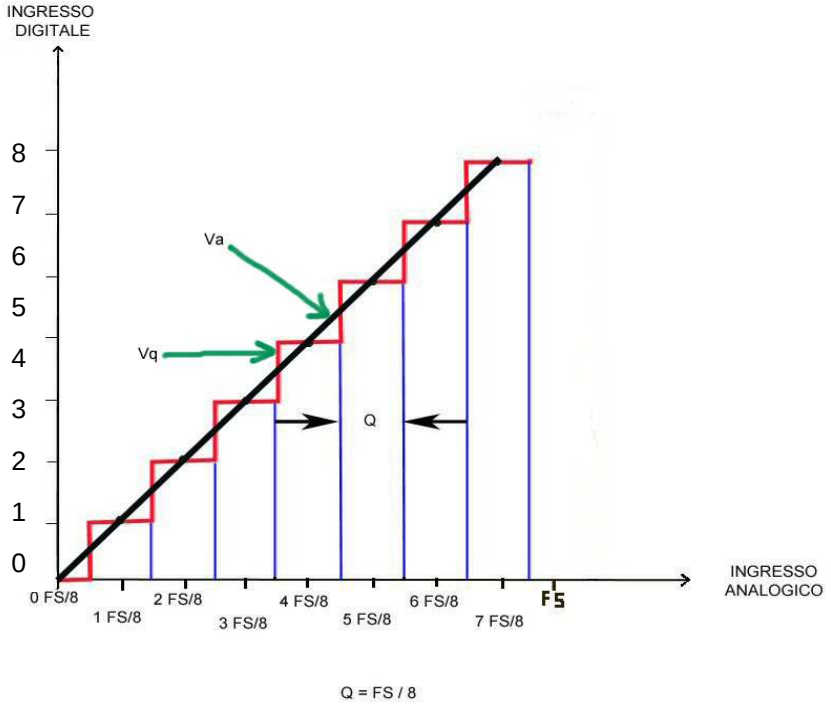
# Quantizzazione

•Ogni campione deve essere rappresentato numericamente da un valore in un insieme di cardinalità finita: Il valore numerico di un campione è rappresentato uno tra N valori (livelli di quantizzazione)

•Gli intervalli di quantizzazione (Q) vengono scelti in funzione delle caratteristiche del segnale da rappresentare.

- Quantizzatore uniforme: Q=FS/N
  dove FS è il Fondo Scala FS=Max-Min

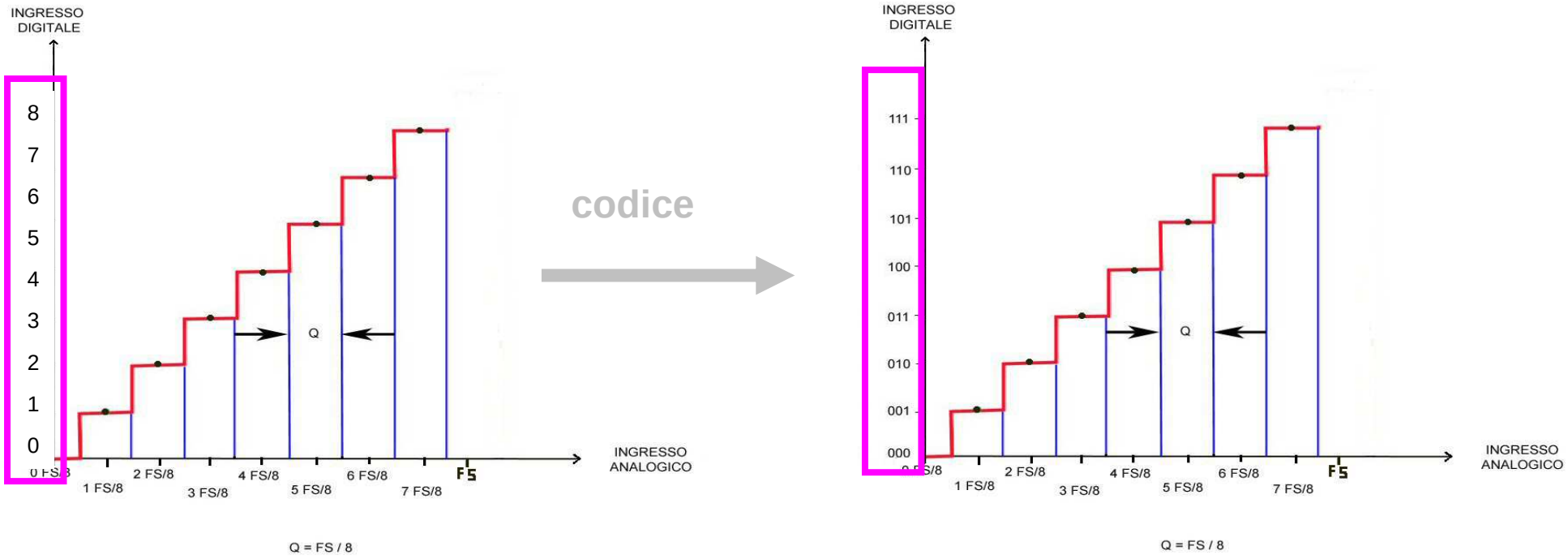



L'errore di quantizzazione è definito come la differenza tra due valori numerici successivi.

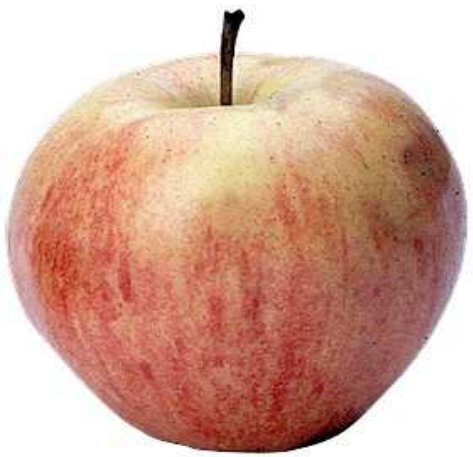
# Codifica

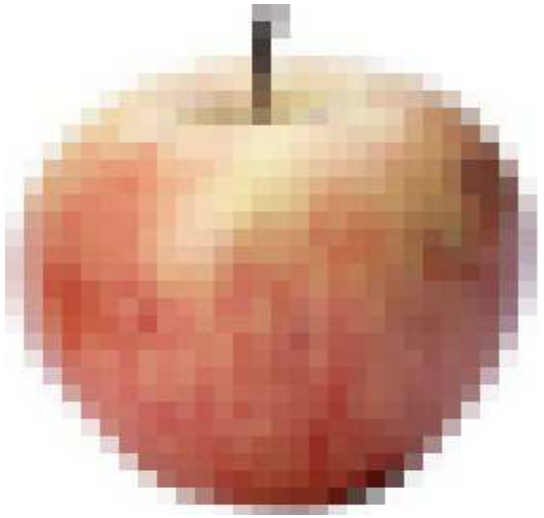
- L’operazione di codifica trasforma i valori numerici forniti dal quantizzatore in valori cifre binarie




la scelta del codice dipende da tipo di informazione che vogliamo rappresentare

# Rappresentazione Discreta di Informazioni Continue: le Immagini





I quadratini della griglia sono chiamati ***pixel*** (picture elements) unità costituenti dell'immagine

Il numero di pixel in cui è suddivisa un immagine si chiama risoluzione e si esprime con una coppia di numeri ad es. 640 x 480 pixel (orizzontali per verticali)

- Nel caso delle immagini non è presente la dimensione temporale
- Sono presenti due dimensioni spaziali.

L'immagine viene suddivisa in un insieme di piccoli quadratini e viene memorizzata l'informazione relativa al colore presente nel quadratino. Questo procedimento porta ad immagini ***raster*** (dal latino *rastrum*, *rastrello*) ad indicare le righe orizzontali dei televisori o dei monitor

# Immagini in bianco e nero

- Se per ogni pixel viene misurato un bit che codifica l'informazione presente non presente abbiamo una immagine binaria(bianco/nero) se invece associo ad ogni pixel il livello medio di intensità luminosa ho una rappresentazione a livello di grigio dell'immagine
  - ogni pixel è codificato con un numero di bit > 1
- Es.
  - se utilizziamo quattro bit possiamo rappresentare $2^4$=16 livelli di grigio,
  - se utilizziamo otto bit ne possiamo distinguere $2^8$=256

# La codifica del colore

La rappresentazione di un'immagine mediante la codifica diretta dei pixel, viene chiamata bitmap.

Il numero di byte memorizzati per un immagine dipende dalla risoluzione e dal numero di colori che ogni pixel può assumere.

Un possibile modello di rappresentazione è noto con il nome di RGB (Red, Green, Blue), il quale usa questi 3 colori per rappresentare tutti i possibili colori

Nella codifica RGB ogni pixel è rappresentato da una combinazione di 3 numeri, ognuno rappresentante una diversa gradazione di uno dei colori primari.

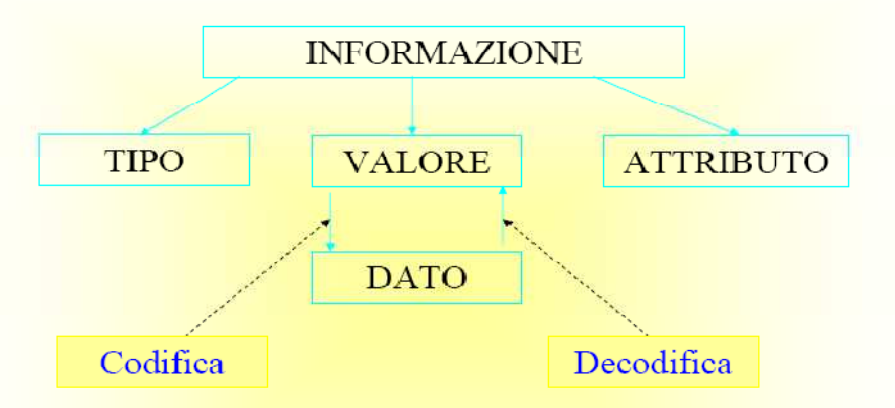
Es.: Con 8 bit per colore otteniamo:
256 x 256 x 256 = 16.777.216 colori diversi
Per ogni pixel sono richiesti 3 byte

Il sistema visivo umano HVS è più sensibile alla luminanza (brightness) che al colore stesso cioè più sensibile ai contorni che alle sfumature di colore;
La rappresentazione RGB attribuisce lo stesso peso alle tre componenti di colore, anche se l'occhio umano è più sensibile alla lunghezza d'onda corrispondente al verde; Una codifica del colore più adatta al HVS è la YCbCr che rappresenta la luminosità Y con più dettaglio (utilizzando più bits) rispetto alle componenti di crominanza (Cb e Cr);

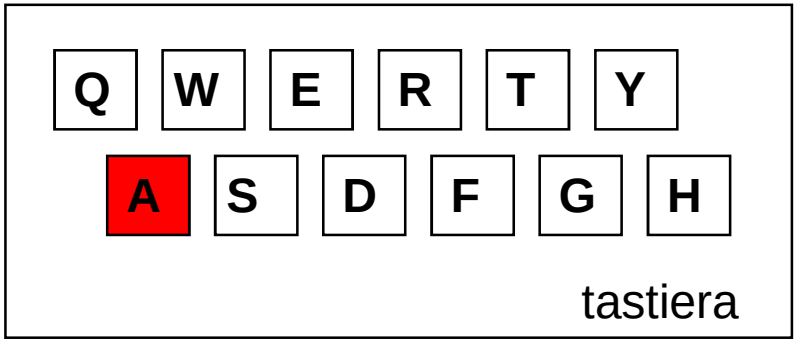
# Torniamo alle informazioni

- Suoni, Immagini, video ed altro sono informazioni di diverso tipo, elemento per elemento assumono valore diverso ed hanno un diverso significato:
- Le informazioni sono caratterizzate dalla tripla {TIPO, VALORE , ATTRIBUTO}
- Quando voglio memorizzare il contatto msn di un amico lo faccio attraverso il suo nome:
  - Tipo = stringa ( vettore di caratteri)
  - Attributo = nome di persona
  - Valore = Pietro

- La relazione tra informazione e dato viene identificata da tutti e tre questi elementi



# Rappresentazione dei caratteri: la codifica ASCII




codice ASCII
della lettera ‘A’

**01000001**

- La Codifica ASCII serve a codificare i caratteri alfanumerici.
- È un sistema di codifica dei caratteri a 7 bit.
- Estensione ad 8 bit per raddoppiare il numero di caratteri rappresentabili (extended ASCII). In questo caso sono codificati simboli speciali per i diversi alfabeti quali lettere greche, lettere accentate, etc.

# La Codifica ASCII: codifica tabellare

**ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange):

| | |
|---|---|
| 0 | 0011 0000 |
| 1 | 0011 0001 |
| 2 | 0011 0010 |
| 3 | 0011 0011 |
| 4 | 0011 0100 |
| 5 | 0011 0101 |
| 6 | 0011 0110 |
| 7 | 0011 0111 |
| 8 | 0011 1000 |
| 9 | 0011 1001 |
| : | 0011 1010 |
| ; | 0011 1011 |
| < | 0011 1100 |
| = | 0011 1101 |
| > | 0011 1110 |
| ? | 0011 1111 |
| @ | 0100 0000 |
| A | 0100 0001 |
| B | 0100 0010 |
| C | 0100 0011 |
| D | 0100 0100 |

| | |
|---|---|
| E | 0100 0101 |
| F | 0100 0110 |
| G | 0100 0111 |
| H | 0100 1000 |
| I | 0100 1001 |
| J | 0100 1010 |
| K | 0100 1011 |
| L | 0100 1100 |
| M | 0100 1101 |
| N | 0100 1110 |
| O | 0100 1111 |
| P | 0101 0000 |
| Q | 0101 0001 |
| R | 0101 0010 |
| S | 0101 0011 |
| T | 0101 0100 |
| U | 0101 0101 |
| V | 0101 0110 |
| W | 0101 0111 |
| X | 0101 1000 |
| Y | 0101 1001 |
| Z | 0101 1010 |

Come sequenza di codifiche ASCII la parola “SALVE” sarà:

01010011 01000001 01001100 01010110 0100101

Per la decodifica si decompone la sequenza di bit in gruppi di byte e si determina il carattere corrispondente ad ogni gruppo di 8 bit.

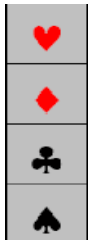
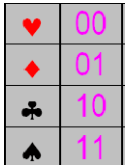
# La Codifica Unicode

- E’ un sistema di codifica che assegna un numero (o meglio, una combinazione di bit) a ogni carattere in maniera indipendente dal programma, piattaforma e dalla lingua (e dal suo sistema di scrittura).
- E’ compatibile col codice ASCII.
- Attualmente codifica i simboli di quasi tutti gli alfabeti moderni.
- Nato come codice a 16 bit (corrispondenti a 65536 caratteri diversi). Per permettere la rappresentazione della totalità dei caratteri si basa su tre tipi di codifiche diverse: UTF-8 a 8 bit (1 byte), UTF-16 a 16 bit (word) e UTF-32 a 32 bit (double word).

# Rappresentazione dei numeri

- Concetti da chiarire:
  - Cardinalità di un insieme
    - è il numero totale di elementi da rappresentare :

      l'insieme P = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}  ha cardinalità  |P| = 16

    - l'insieme Q =  ha cardinalità |Q|= 4
  - Base della rappresentazione
    - Gli esseri umani contano in base 10 (probabilmente deriva dal numero delle dita)
    - I computer in base 2 (bit)
    - Quanti bit servono per memorizzare informazioni relative ai due insiemi?
    - n = log2 ( | . | )
    - $n_Q$= 2 corrispondenti a 

      caratteri

      $n_P$= 4 corrispondenti a …??? Abbiamo bisogno di un metodo di codifica più efficiente di quello tabellare visto con i

# Rappresentazione dei numeri

- Tra i primi sistemi di numerazione abbiamo quello romano:
  - Cifre →
  - Sistema addizionale poco adatto per la rappresentazione di numeri molto grandi
  - Non era rappresentato lo zero
  - Operazioni complesse

- Base di numerazione araba
  - **Cifre: 0 1 2 3 4 5 6 7 8 9**
  - È una rappresentazione posizionale
  - possibile per la presenza dello zero
  - Operazioni più semplici

Esempio:

3201 =

$(3x10^3) + (2x10^2) + (0x10^1) + (1x10^0)$

| | | | |
|---|---|---|---|
| 1 | I | 30 | XXX |
| 2 | II | 40 | XL |
| 3 | III | 50 | L |
| 4 | IIII (IV) | 60 | LX |
| 5 | V | 70 | LXX |
| 6 | VI | 80 | LXXX |
| 7 | VII | 90 | XC |
| 8 | VIII | 100 | C |
| 9 | IX (VIIII) | 200 | CC |
| 10 | X | 300 | CCC |
| 11 | XI | 400 | CD |
| 12 | XII | 500 | D (IↃ) |
| 13 | XIII | 600 | DC |
| 14 | XIV | 700 | DCC |
| 15 | XV | 800 | DCCC |
| 16 | XVI | 900 | CM |
| 17 | XVII | 1.000 | M (CIↃ) |
| 18 | XVIII | 2.000 | MM, CIↃCIↃ, $\overline{\text{II}}$ |
| 19 | XIX | 10.000 | CCIↃↃ, $\overline{\text{X}}$ |
| 20 | XX | 100.000 | CCCIↃↃↃ, $\overline{\text{C}}$ |
| | | 1.000.000 | CCCCIↃↃↃↃ, $\overline{\text{\|X\|}}$ |

# In generale

- Rappresentazione in base B → B-1 cifre
  - 0 1 2 … B-1
    - $d_i$ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } in base 10
    - $d_i$ = { 0, 1 } in base 2
    - $d_i$ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F } in base 16

- Conversione verso la base decimale:
  - $d_{31}d_{30} \ldots d_2d_1d_0$ è un numero a 32 cifre
  - valore =
    $d_{31} \times B^{31} + d_{30} \times B^{30} + \ldots + d_2 \times B^2 + d_1 \times B^1 + d_0 \times B^0$

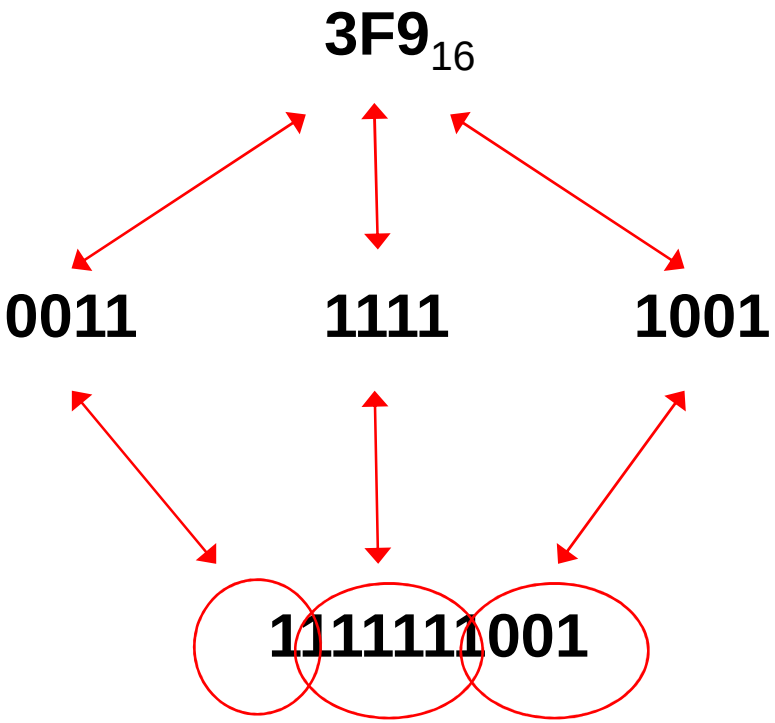
# Esempio di conversione in base 10

- da B=2 :
  - cifre: 0 1
  - 1011010 →
    $1x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 1x2 + 0x1 =$
    $64 + 16 + 8 + 2 = 90$ 7 cifre binarie → 2 cifre decimali

- da B=16 :
  - cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - 524 →
    $5x16^2 + 2x16 + 4x1 = 1316$
    3 cifre esadecimali → 4 cifre decimali

# Conversione esadecimale-binario

Siccome $16=2^4$, il passaggio tra le rappresentazioni in base 2 e in base 16 è molto semplice:




| base | | |
|---|---|---|
| 10 | 16 | 2 |
| 00 | 0 | 0000 |
| 01 | 1 | 0001 |
| 02 | 2 | 0010 |
| 03 | 3 | 0011 |
| 04 | 4 | 0100 |
| 05 | 5 | 0101 |
| 06 | 6 | 0110 |
| 07 | 7 | 0111 |
| 08 | 8 | 1000 |
| 09 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

# Quale base usare ?

- Decimale
  - naturale per gli esseri umani.
- Esadecimale
  - utile (agli esseri umani) per esaminare lunghe stringhe di bit
- Binaria
  - rappresentazione ottimale per il calcolatore

… perché non usare una codifica binaria della rappresentazione in base 10 ?

# Conversione base 10 → base 2 (interi)

Come ottenere la rappresentazione in base 2 di un numero intero T rappresentato in base 10 ?

**Supponiamo:**

$$\mathbf{T=c_{n-1}x2^{n-1} + c_{n-2}x2^{n-2} + \ldots + c_2x2^2 + c_1x2^1 + c_0x2^0}$$

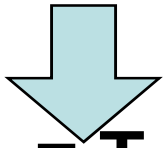
$$\mathbf{c_i \in \{0,1\}}$$

**Non conosciamo:**

- le cifre $c_i$
- il numero di cifre n

# Conversione base 10 → base 2 (interi)

$$T=c_{n-1}x2^{n-1} + c_{n-2}x2^{n-2} + \ldots + c_2x2^2 + c_1x2^1 + c_0x2^0 =$$
$$(c_{n-1}x2^{n-2} + c_{n-2}x2^{n-3} + \ldots + c_2x2^1 + c_1)\,x2 + c_0 =$$
$$Q_0x2 + c_0$$



$$Q_0 = T \text{ div } 2 \qquad c_0 = T \bmod 2$$

$$Q_0 = (c_{n-1}x2^{n-3} + c_{n-2}x2^{n-4} + \ldots + c_2)x2 + c_1 = Q_1x2 + c_1$$
$$Q_1 = Q_0 \text{ div } 2 \qquad c_1 = Q_0 \bmod 2$$

# Conversione base 10 → base 2 (interi)

- Non si conoscono ne le cifre ne il numero di cifre.
- Si divide il numero per la base e si prende il resto in ordine inverso.

- Es.:
- 43/2 = 21 con resto 1
- 21/2 = 10 con resto 1
- 10/2 = 5 con resto 0
- 5/2 = 2 con resto 1
- 2/2 = 1 con resto 0

$43_{10} = 101011_2$

# Conversione base 10 → base 2 (interi)

```
void convint(int T,int c[],int &n)
{
  int Q;
  n=0;Q=T;
  do {
   c[n]=Q%2;
   Q=Q/2;
   n++;
  } while (Q!=0);
}
```

La conversione genera le cifre a partire da quella meno significativa

Esempio:
$75_{10}$ → **?**$_2$

# Aritmetica in base 2

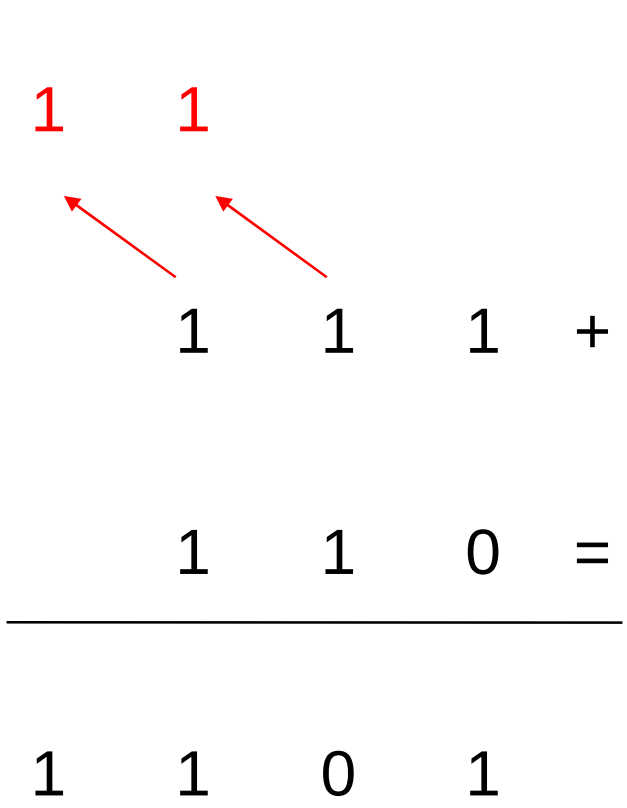
Le operazioni aritmetiche si svolgono in maniera analoga a quanto si fa in base 10.

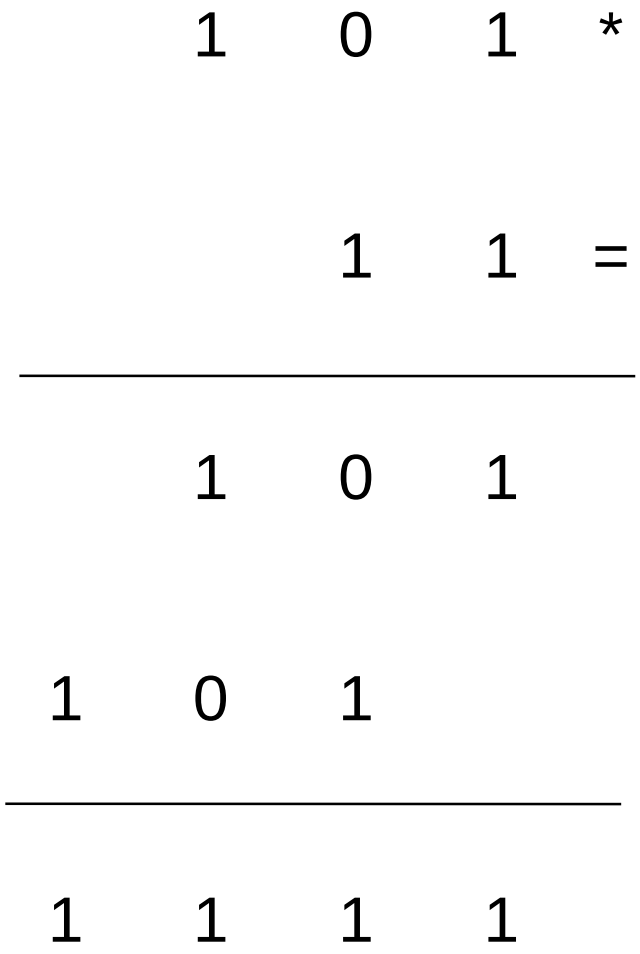
| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

| * | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

"tavola pitagorica" in base 2

# Aritmetica in base 2



# Aritmetica dei registri

- I registri di memoria sono supporti di lunghezza finita
- Ciò impone delle restrizioni all'insieme di numeri rappresentabili e, di conseguenza, dei vincoli all'aritmetica
- Registro a N bit → $2^N$ valori diversi rappresentabili
  - Es.: 8 bit → 256 valori
    possibile rappresentare l'intervallo [0,255]

# Aritmetica dei registri

**Non ci sono problemi nel caso in cui l'operazione produce un risultato rappresentabile nel registro**

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | + | | 1 | 0 | 9 | + |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | = | | 1 | 3 | 7 | = |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | | 2 | 4 | 6 | |

# Aritmetica dei registri

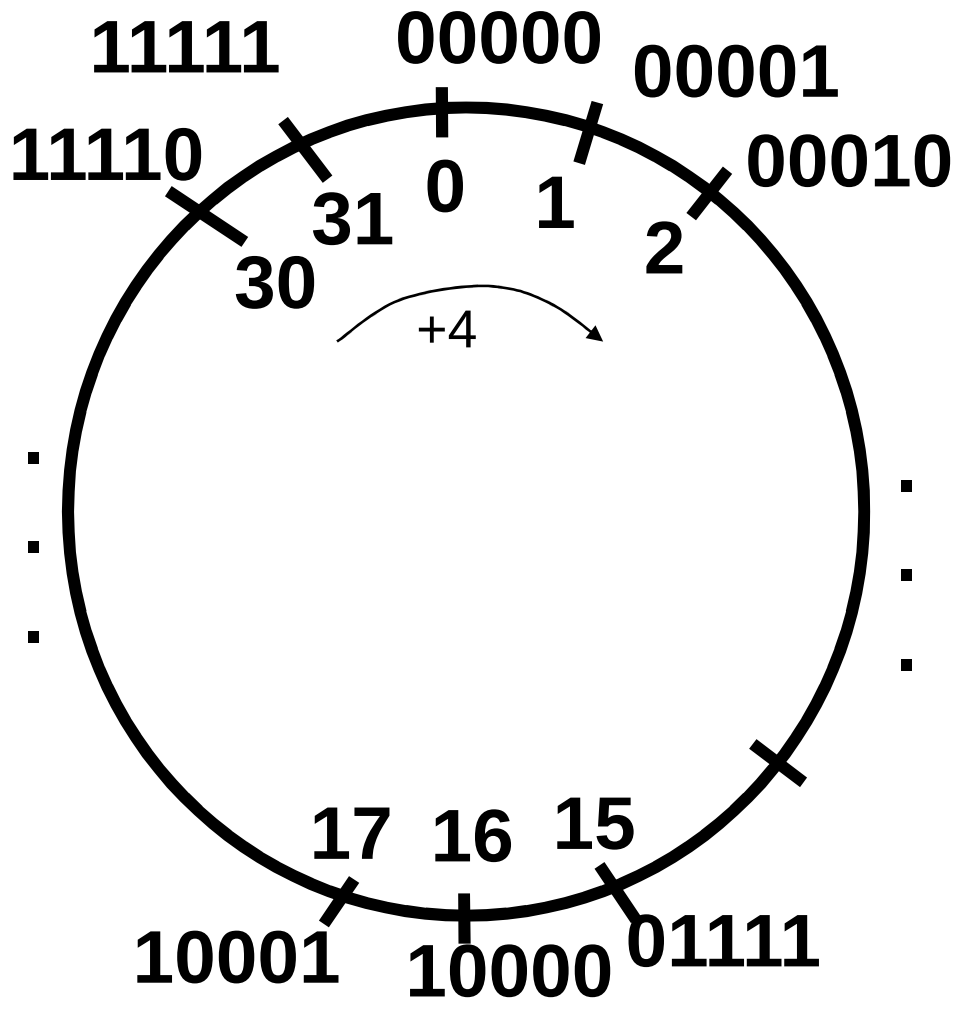
**Se l'operazione fornisce un risultato R non rappresentabile, si produce un riporto uscente dal registro, mentre all'interno rimane una parte della rappresentazione del risultato (R mod $2^N$)**

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | + | 2 | 3 | 7 | + |
| | | | | | | | | | | 1 | 3 | 7 | = |
| ~~1~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~1~~ | ~~0~~ | ~~0~~ | ~~1~~ | ~~=~~ | | | | | |
| | | | | | | | | | | 3 | 7 | 4 | |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | | | | |
| | | | | | | | | | | 1 | 1 | 8 | |

# Aritmetica dei registri

- L'aritmetica dei registri a N bit è caratterizzata da una congruenza mod $2^N$

- Quindi, per N=5:
  - 30+4=2 !



# Aritmetica dei registri

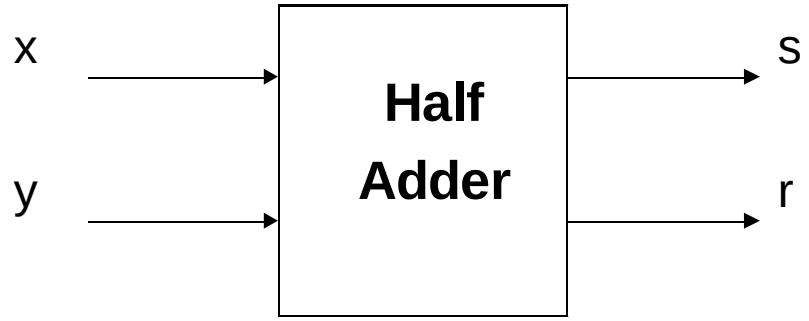
- Il riporto uscente dal registro, generato da un'addizione tra numeri interi, si definisce *carry*
- Il prestito uscente dal registro, generato da una sottrazione tra numeri interi, si definisce *borrow*

| | | | | |
|---|---|---|---|---|
| 4 | 00100 | 10 | 01010 | |
| + 2 | + 00010 | + 26 | 11010 | |
| 6 | 0\|00110 | 4 | 1\|00100 | **carry** |

| | | | | |
|---|---|---|---|---|
| 4 | 00100 | 10 | 01010 | |
| - 2 | - 00010 | - 26 | 11010 | |
| 2 | 0\|00010 | 16 | 1\|10000 | **borrow** |

# Struttura logica dei circuiti per l'addizione

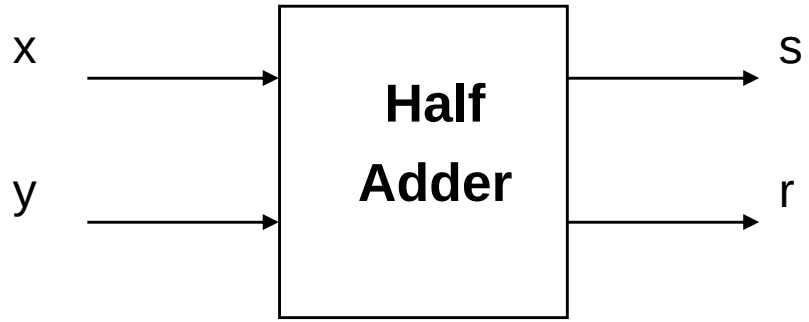
- Come sarà fatto un circuito per l'addizione tra registri ?
- Partiamo da un circuito per l'addizione di cifre binarie.



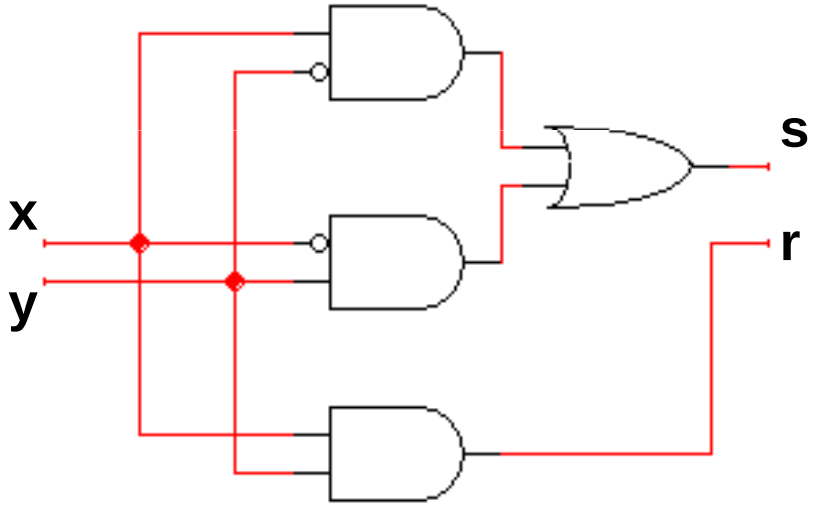

**Half Adder (HA)**

| x | y | s | r |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

# Half Adder (HA)



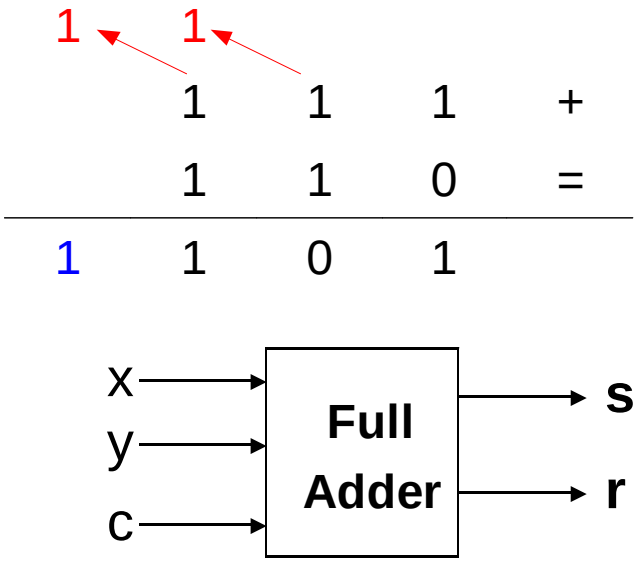

| x | y | s | r |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



# Struttura logica dei circuiti per l'addizione

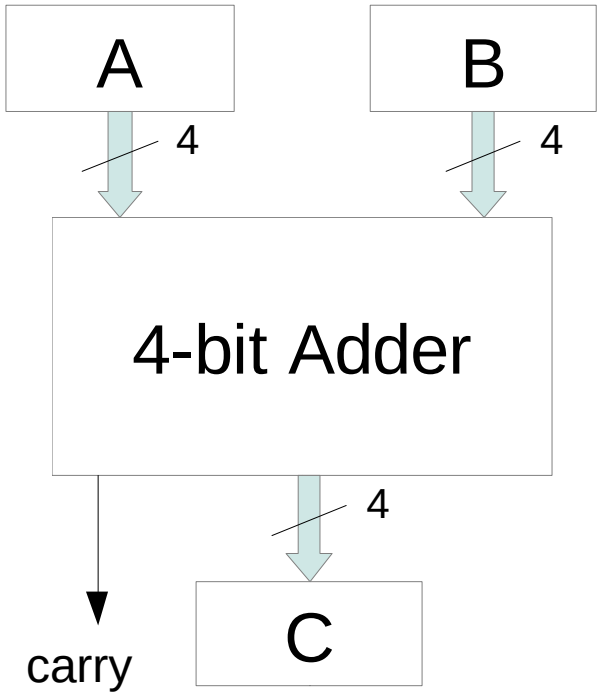
- Di fatto il solo Half Adder non basta perchè, in presenza di un riporto entrante, sarebbe necessario sommare tre cifre binarie

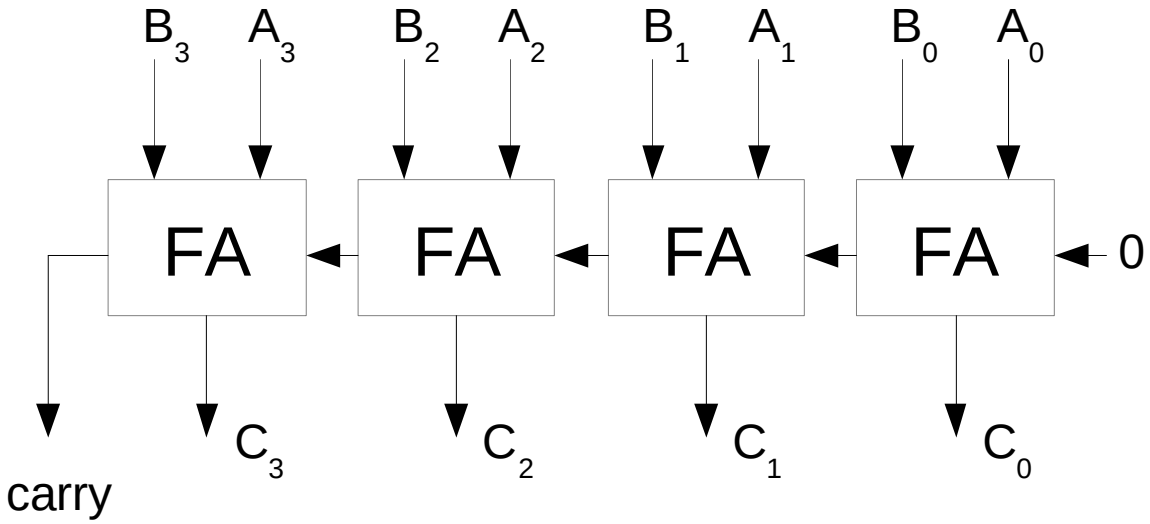
| x | y | c | s | r |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |




**Full Adder (FA)**

# Struttura logica dei circuiti per l'addizione



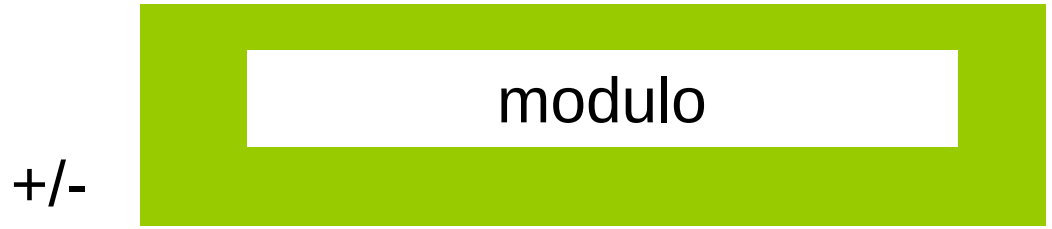



# Rappresentazione dei numeri relativi

- Rappresentazione in segno e modulo
- Rappresentazione in complementi alla base
- Rappresentazione per eccessi

# Rappresentazione dei numeri negativi

- Soluzione più immediata: segno + modulo




Possibile convenzione:

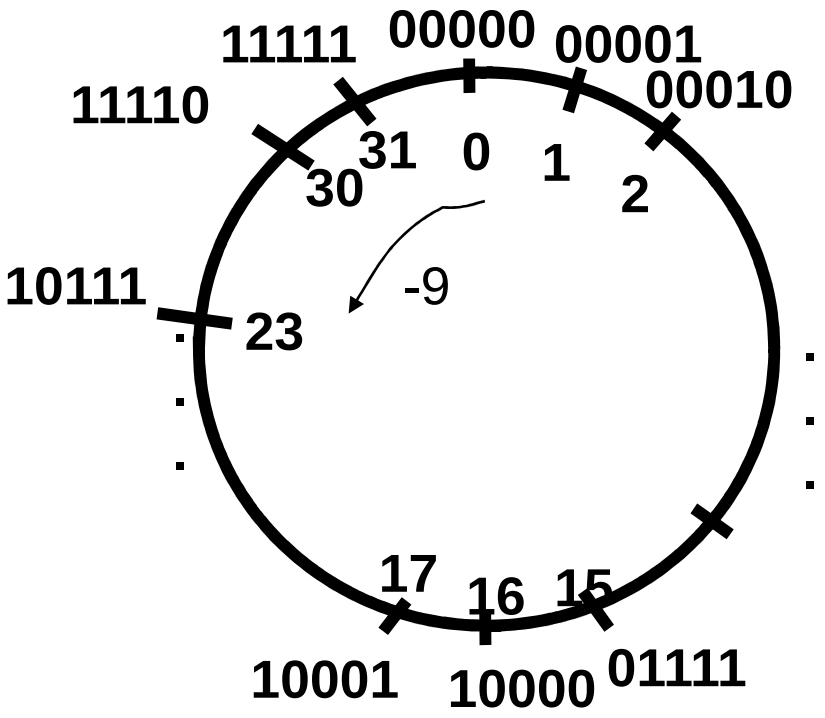
0 → +  1 → -

- Problemi
  - dove mettere il segno ?
  - doppia rappresentazione per lo zero (+0, -0)
  - operazioni alquanto complicate

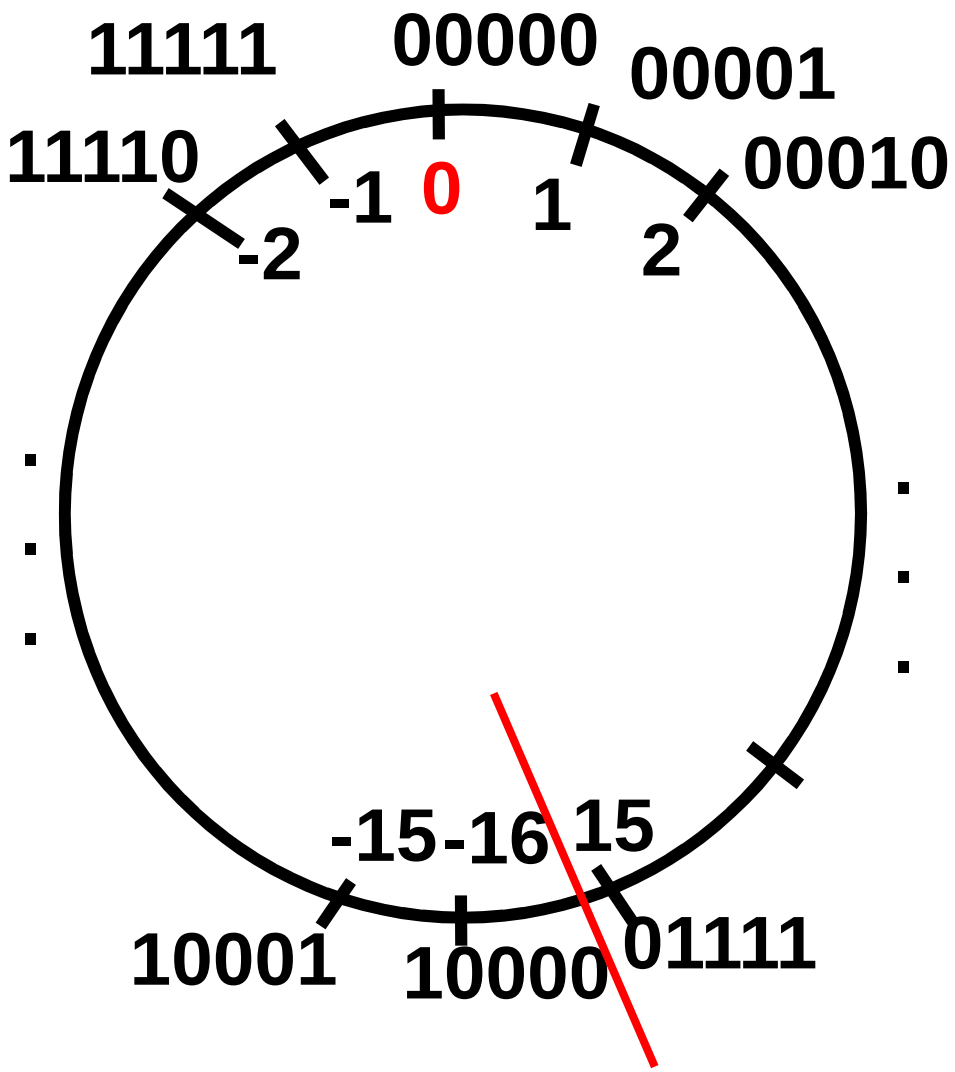
# Rappresentazione dei numeri negativi

- Soluzione alternativa
  - Che cosa succede in un registro a N bit quando si sottrae un numero da 0 ?

$$
\begin{array}{r}
0\ 0\ 0\ 0\ 0\ - \\
\hline
0\ 1\ 0\ 0\ 1\ = \\
\\
1\ 1\ 0\ 1\ 1\ 1
\end{array}
$$



# Complementi alla base




**Caratteristiche:**

- $2^{N-1}$ non-negativi
- $2^{N-1}$ negativi
- uno zero
- quanti positivi ?
- confronto ?
- rappr. dello zero

# Complementi alla base

- L'intervallo di numeri rappresentati è $[-2^{N-1} \; +2^{N-1}-1]$
- La rappresentazione di un numero x nell'intervallo è data da $R(x)=(x+2^N) \bmod 2^N$
- Il bit più significativo è indicativo del segno ("bit di segno")

| | |
|---|---|
| 00000 | 0 |
| 00001 | +1 |
| 00010 | +2 |
| 00011 | +3 |
| . | . |
| . | . |
| 01110 | +14 |
| 01111 | +15 |
| 10000 | -16 |
| 10001 | -15 |
| 10010 | -14 |
| . | . |
| . | . |
| 11101 | -3 |
| 11110 | -2 |
| 11111 | -1 |

# Calcolo rapido del complemento alla base

- Per ottenere rapidamente la rappresentazione in complemento alla base di un numero negativo su N bit
  - si estrae la rappresentazione del valore assoluto del numero su N bit
  - si complementano le cifre ad una ad una
  - si aggiunge 1
- Es.: complemento alla base su 8 bit di -33

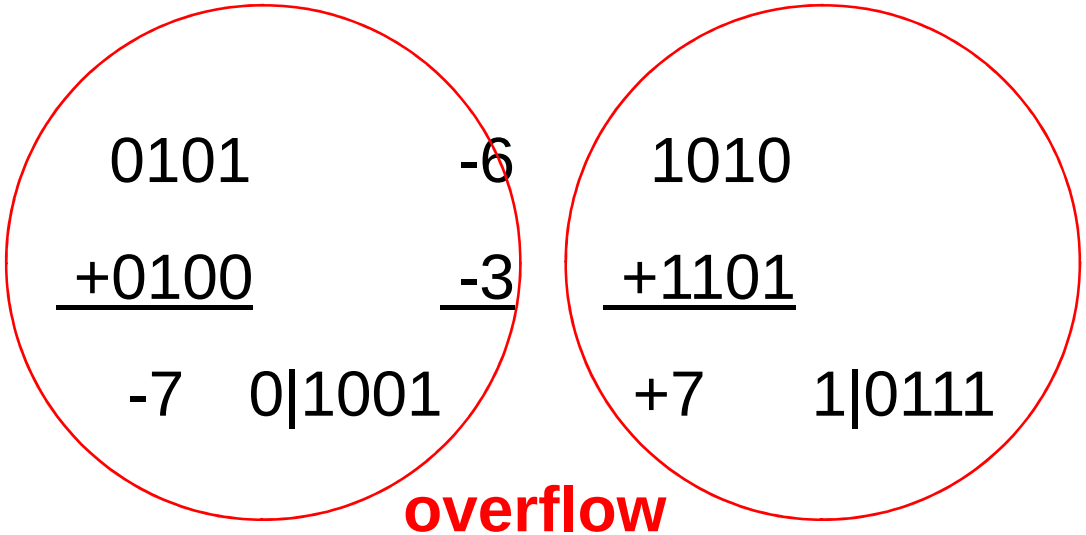
  $33_{10}$ =00100001  11011110+1=11011111

# Operazioni in complemento alla base

- Le addizioni si realizzano direttamente sulle rappresentazioni in quanto R(x+y)=R(x)+R(y)
- Anche le sottrazioni si valutano tramite addizioni, ponendo x-y come x+(-y); di conseguenza R(x-y)=R(x)+R(-y)
- Nel caso in cui l'operazione produce un numero al di fuori dell'intervallo di rappresentazione si ha un *overflow*

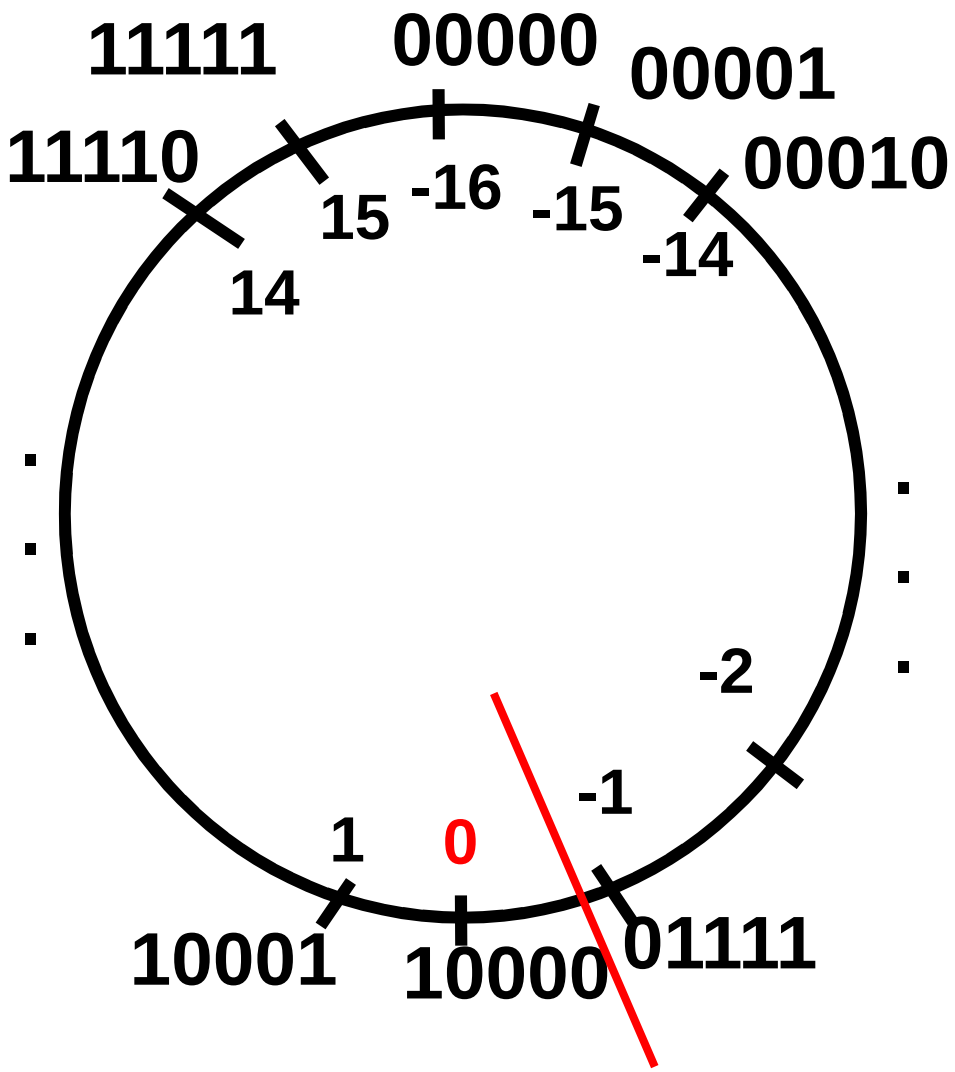
# Operazioni in complemento alla base

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| +4 | 0100 | | | +4 | 0100 | | |
| +2 | + 0010 | | | -2 | +1110 | | |
| +6 | 0\|0110 | | +2 | 1\|0010 | | | |
| -4 | 1100 | +5 | 0101 | -6 | 1010 | | |
| -2 | + 1110 | +4 | +0100 | -3 | +1101 | | |
| -6 | 1\|1010 | | -7 | 0\|1001 | +7 | 1\|0111 | |



**overflow**

# Rappresentazione per eccessi (polarizzata)




**Caratteristiche:**

- $2^{N-1}$ non-negativi
- $2^{N-1}$ negativi
- uno zero
- quanti positivi ?
- confronto ?
- rappr. dello zero

# Eccessi

- L'intervallo di numeri rappresentati è $[-2^{N-1} \; +2^{N-1}-1]$
- La rappresentazione di un numero x nell'intervallo è data da $R(x)=x+2^{N-1}$
- Il bit più significativo è indicativo del segno ("bit di segno")

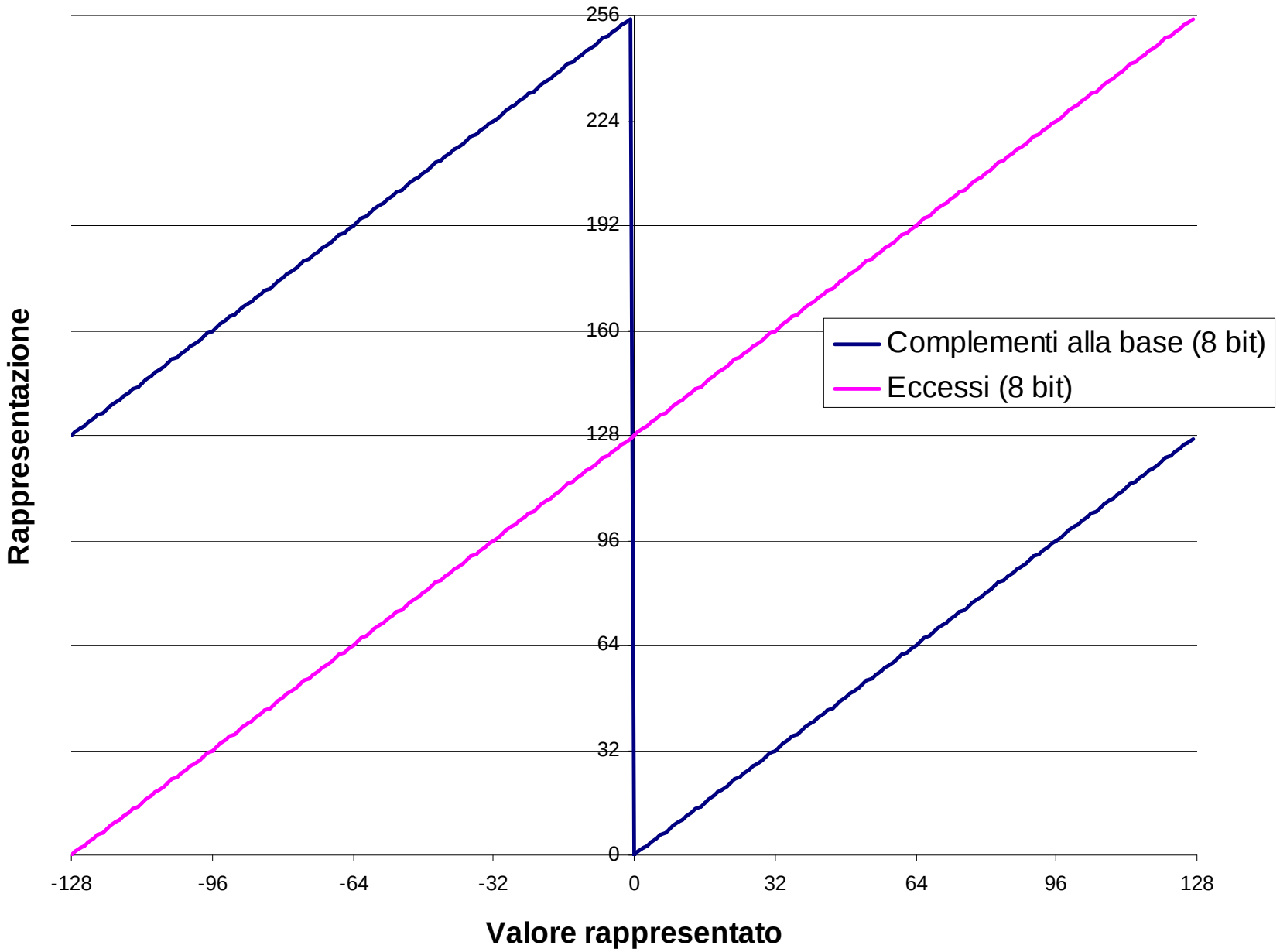
| | |
|---|---|
| 00000 | -16 |
| 00001 | -15 |
| 00010 | -14 |
| 00011 | -13 |
| . | . |
| . | . |
| 01110 | -2 |
| 01111 | -1 |
| 10000 | 0 |
| 10001 | +1 |
| 10010 | +2 |
| . | |
| . | . |
| 11101 | +13 |
| 11110 | +14 |
| 11111 | +15 |

# Operazioni in eccessi

- Le addizioni si realizzano direttamente sulle rappresentazioni in quanto R(x+y)=R(x)+R(y)
- Anche le sottrazioni si valutano tramite addizioni, ponendo x-y come x+(-y); di conseguenza R(x-y)=R(x)+R(-y)
- Achtung! Siccome $R(x)+R(y)=x+y+2^{N-1}+2^{N-1}$, il risultato necessita di una correzione
- Nel caso in cui l'operazione produce un numero al di fuori dell'intervallo di rappresentazione si ha un *overflow*

# Confronto tra complementi alla base ed eccessi

- Entrambe permettono di realizzare una sottrazione tramite addizione (macchine aritmetiche più semplici)
- Le operazioni in eccessi richiedono un aggiustamento finale
- La rappresentazione in complementi rende più difficile il confronto


256
224
192
160
128
96
64
32
0
Rappresentazione
-128
-96
-64
-32
0
32
64
96
128
Valore rappresentato
Complementi alla base (8 bit)
Eccessi (8 bit)

# Numeri signed e unsigned

- Un registro di N bit può rappresentare:
  - Numeri assoluti nel range **[0, $2^N$-1]** → numeri ***unsigned***
  - Numeri relativi nel range **[-$2^{N-1}$, $2^{N-1}$-1]** → numeri ***signed***

} **C**

- Dalla stringa di bit nel registro non si può risalire al tipo di numero memorizzato. Quali sono le conseguenze ?
  - Operazioni aritmetiche indipendenti dalla rappresentazione → nessuna conseguenza
  - Confronto dipendente dalla rappresentazione → due tipi di confonto

- X = 10001 Y = 01110

  X > Y ?
  - unsigned: SI (17>14)
  - signed: NO (-15<+14)