Rappresentazione dei dati

Rappresentazione in base 2 e base 16
Aritmetica dei registri

Come rappresentiamo i numeri ?

- **Base di numerazione: dieci**
  - **Cifre: 0 1 2 3 4 5 6 7 8 9**
- **Rappresentazione posizionale**
  - **possibile per la presenza dello zero**

Esempio:

3201 =

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

In generale …

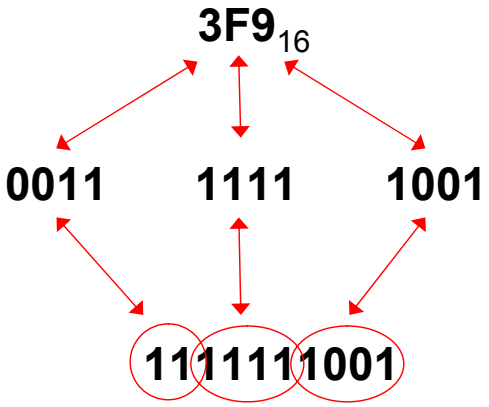
- **Rappresentazione in base B → B-1 cifre**
  - 0 1 2 … B-1

- **Rappresentazione dei numeri:**
  - $d_{31}d_{30}$ ... $d_2d_1d_0$ è un numero a 32 cifre
  - valore =
    $d_{31} \times B^{31} + d_{30} \times B^{30} + ... + d_2 \times B^2 + d_1 \times B^1 + d_0 \times B^0$

Altre basi

- **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
- **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

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:

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

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

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)

```
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:

$75_{10} \rightarrow ?_2$

Conversione base 10 → base 2 (frazionari)

Consideriamo un numero F minore di 1.

$F = c_{-1}x2^{-1} + c_{-2}x2^{-2} + \ldots + c_{-n}x2^{-n}$ $\quad c_i \in \{0,1\}$

$Fx2 = c_{-1} + (c_{-2}x2^{-1} + \ldots + c_{-n}x2^{-(n-1)}) = c_{-1} + P_1$ $\quad P_1<1$

$P_1x2 = c_{-2} + (c_{-3}x2^{-1} + \ldots + c_{-n}x2^{-(n-2)}) = c_{-2} + P_2$

Conversione base 10 → base 2 (frazionari)

```
void convfrac(float F,int c[],int &n)
{
  float P;
  n=0;P=F;
  do {
     c[n]=(int)(P*2);
     P=P*2-c[n];
     n++;
  } while (P==0 || n>=NMAX);
}
```

La conversione genera le cifre a partire da quella più significativa

Esempio:

0.625_{10} → $?_2$

Aritmetica in base 2

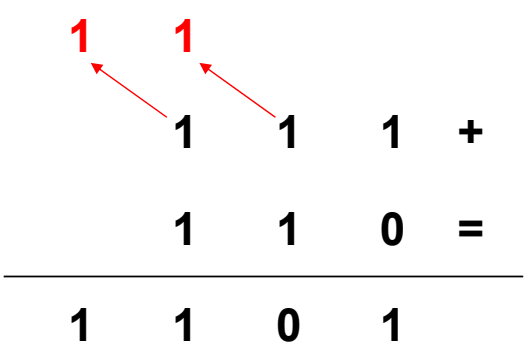
Le operazioni aritmetiche si svolgono in maniera analoga a quanto si fa in base 10.

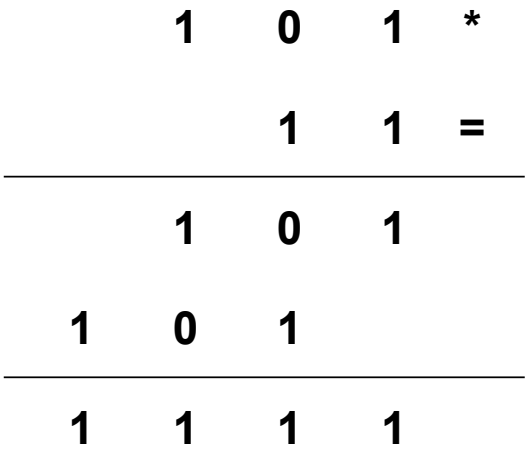
+	0	1
0	0	1
1	1	10

*	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	

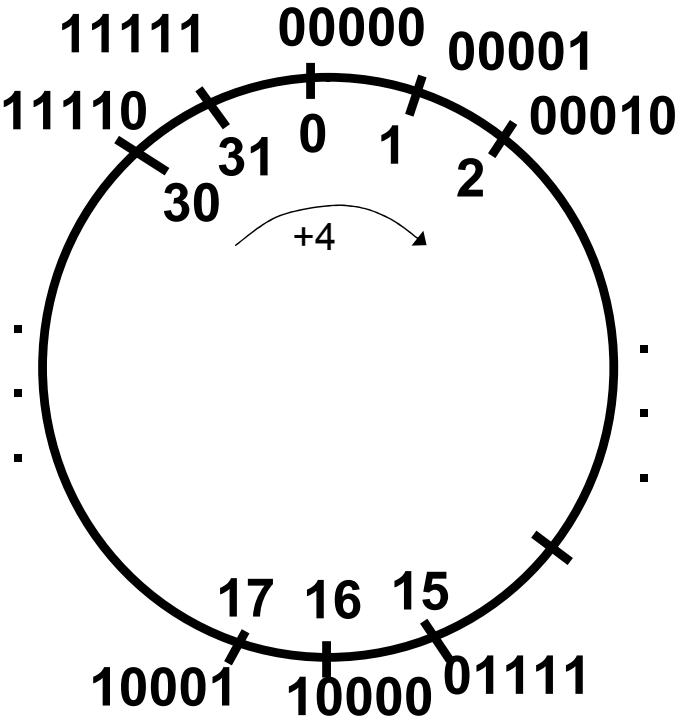
Aritmetica dei registri

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	0	0	0	1	0	0	1	=		1	3	7	=
1	0	1	1	1	0	1	1	0			3	7	4	
											1	1	8	

Aritmetica dei registri

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