
Sistemi di numerazione binaria

Numeri e numerali

Numero: *concetto astratto*

Numerale : *stringa di caratteri che rappresenta un numero in un dato sistema di numerazione*

- Lo stesso numero è rappresentato da numerali diversi in diversi sistemi

ESEMPIO

- 156 nel sistema decimale
- CLVI in numeri romani

- Il numero di caratteri nel numerale determina l'intervallo di numeri rappresentabili

ESEMPIO

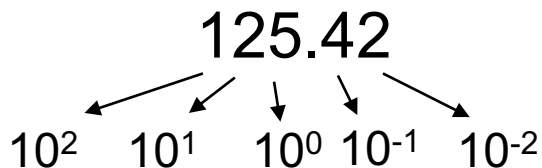
- Interi a 3 cifre con segno in notazione decimale: [-999,+999]

Sistemi posizionali

*Il numero è rappresentato come somma di potenze della base **b**, ciascuna moltiplicata per un coefficiente intero*

$a_m a_{m-1} \dots a_0 . a_{-1} a_{-2} \dots a_{-k}$

$$N = \sum_{i=-k}^m a_i b^i \quad \begin{array}{l} 0 \leq a_i \leq b-1 \\ b = \text{base} \end{array}$$



- Ciascuna cifra del numerale rappresenta il coefficiente di una potenza della base
- L'esponente è dato dalla *posizione* della cifra

Aritmetica e notazione posizionale

- Lo scarso sviluppo dell'aritmetica in età classica è legato all'uso di una notazione non posizionale

La notazione posizionale consente di effettuare le operazioni aritmetiche operando sui numerali

- Ad esempio, per sommare due numeri:
 - Si incolonnano i numerali
 - Si sommano cifre omologhe (coefficienti della stessa potenza della base)
 - Si propagano i riporti
 - Si ottiene il numerale che rappresenta la somma
- Invece nella notazione romana.....

Simboli per le cifre

- Se la base è b occorrono b simboli per rappresentare le cifre del numerale
- Per la base 10 utilizziamo le cifre arabe
- Per basi inferiori a 10 se ne usa un sottoinsieme
- Per basi superiori a 10 occorre definire simboli aggiuntivi
- Casi di interesse:

$b = 10 \{0,9\}$

$b = 2 \{0,1\}$

$b = 8 \{0,1, \dots 7\}$

$b = 16 \{0,1, \dots 9, A, B, C, D, E, F\}$

Notazione decimale e binaria

- Sono entrambe notazioni posizionali con basi rispettivamente pari a **10** e **2**

Es

$$(5487)_{10} = 7 \cdot 10^0 + 8 \cdot 10^1 + 4 \cdot 10^2 + 5 \cdot 10^3$$

$$(10110)_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = (22)_{10}$$

- Notare come nelle due notazioni occorre un diverso numero di cifre per rappresentare lo stesso numero
- In base 2 occorre circa il triplo delle cifre rispetto alla base 10
- Del resto per rappresentare $(10)_{10}$ in base 2 occorrono 4 cifre:

$$(10)_{10} = (1010)_2$$

$$\begin{array}{ccc} & (10.1)_2 & \\ \swarrow & \downarrow & \searrow \\ 2^1 & 2^0 & 2^{-1} \\ & \Downarrow & \\ & (2.5)_{10} & \end{array}$$

Ordini di grandezza binari

- In un sistema binario gli ordini di grandezza sono dati dalle potenze di 2

$2^0 \dots 2^9 = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512..$

$2^{10} = 1024 \qquad \sim 10^3 \quad 1K$

$2^{20} = 2^{10} 2^{10} = 1048576 \qquad \sim 10^6 \quad 1M$

$2^{30} = 2^{10} 2^{10} 2^{10} = 1073741824 \qquad \sim 10^9 \quad 1G$

$2^{40} = \dots = 1099511627770 \qquad \sim 10^{12} \quad 1T$

ES

$2^{26} = 2^6 \cdot 2^{20} = 64 M$

Notazione in base 16

- Per i numerali esadecimali occorrono 16 cifre

$$\{ 0,1, \dots 9, A,B,C,D,E,F \} \quad (13)_{16} = (19)_{10}$$

- Conversione esadecimale-binario: $00010011 = (19)_{10}$

Si fa corrispondere a ciascuna cifra esadecimale il gruppo di 4 bit che ne rappresenta il valore ($2^4=16$)

ESEMPIO

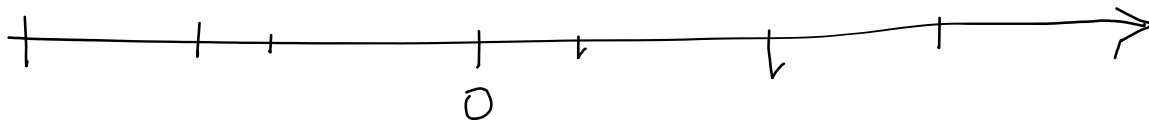
F	5	7	A	3	1
1111	0101	0111	1010	0011	0001

- Conversione binario-esadecimale:

Partendo da destra si sostituisce a ciascun gruppo di 4 o meno cifre binarie la cifra esadecimale che ne rappresenta il valore

Quanti numeri posso rappresentare con 8 bit?

- indipendentemente dalla codifica (a breve) i numeri rappresentabili sono 2^8



Notazione in base 16

- Non è utilizzato per fare conti
 - Non esistono ALU in esadecimale...
- Serve solo agli esseri umani per gestire numerali binari molto lunghi...
- Esempio parola di 32 bit :

01000000111111000000111111010011

0100 0000 1111 1100 0000 1111 1101 0011

4 0 F C 0 F D 3

40FC0FD3

Esempio codice ASCII di 8 bit : $(41)_{16} = (0100\ 0001)_2 = (65)_{10} \rightarrow 'A'$

Tutti gli esempi di somma sono in binario – si omette ()₂

$$1+1 = 10$$

in una somma a più cifre $1+1=0$ col riporto di 1

$$\begin{array}{r} 111111111+ \\ 000000001= \\ \hline 1000000000 \end{array}$$

$$111111111=511 \quad 512-1$$

Ordini di grandezza base 2 base 10

$$2^9=512 \quad 2^9=10^x \quad x=\mathbf{9/10*3}=2.7 \quad 2^9 \sim 10^{2.7}$$

divido l'esponente per 10 e moltiplico per 3

$$2^{10} \sim 10^3$$

Numeri a precisione finita

- Nelle rappresentazioni *con numero finito di cifre* si perdono alcune proprietà:
 - chiusura degli operatori ($+$, $-$, \times)
 - proprietà associativa, distributiva,...

ESEMPIO

- 2 cifre decimali e segno $[-99, +99]$
- $78+36=114$ (*chiusura della somma*)
- $60+(50-40) \neq (60+50)-40$ (*proprietà associativa*)

- Si introducono errori di arrotondamento
- Si introducono buchi nella rappresentazione dei reali

ESEMPIO

- Con numerali decimali con due sole cifre frazionarie non posso rappresentare correttamente 0.015

Numeri a precisione finita

- Non è un puro esercizio matematico
- Può portare a errori incalcolabili
- Quanto fa 8 miliardi + 8 miliardi?

```
1  int main()
2  {
3      int i;
4      i=8000000000+8000000000;
5      printf("i vale : %d",i);
6      return 0;
7  }
```

main

Output

main.c

i vale : -1179869184

```
int main()
{
    int i;
    i=8000000000+8000000000;
    printf("i vale : %d",i);
    return 0;
}
```

Conversione decimale-binario

- Si effettuano divisioni ripetute per 2 sino ad avere il quoziente=0
- Il resto delle divisioni fornisce le cifre del numerale binario (a partire dalla meno significativa)

ESEMPIO $(26)_{10} = (11010)_2$		
26	0	<i>cifra meno significativa</i>
13	1	
6	0	
3	1	
1	1	<i>cifra più significativa</i>
0		

Conversione decimale-binario

(2)

- Altrimenti si può procedere 'a occhio'
- I coefficienti della notazione binaria sono **0** o **1**
- Un numero intero è rappresentabile come somma di un sottoinsieme delle potenze di 2
- Si cerca la più grande potenza di due contenuta nel numero, la si sottrae e si prosegue con la differenza
- Ogni cifra del numerale indica se la corrispondente potenza di due è presente nella somma o no
- **Esempio $26 = 16 + 8 + 2$**

ESEMPIO $(26)_{10} = (11010)_2$

$$(26)_{10} = 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

Intervalli rappresentati

- Rappresentando gli interi positivi e lo zero in notazione binaria con n cifre (bit) si copre l'intervallo $[0, 2^n - 1]$
- Si sfruttano tutte le 2^n disposizioni

<i>ESEMPIO</i>	$n=3$	$[0, 2^3 - 1] \rightarrow [0, 7]$
	0	000
	1	001
	2	010
	3	011
	4	100
	5	101
	6	110
	7	111

NB Anche gli 0 non significativi devono essere rappresentati

Interi positivi e negativi

- Per rappresentare gli interi relativi, a parità di cifre si *dimezza l'intervallo dei valori assoluti*
- Per esempio con **n = 3** bit possiamo rappresentare **8** valori
 - [0 , 7]
 - [-3,4]
 - [-4,3]
 - [-3,+3] (avanza una codifica)
- Si possono usare varie rappresentazioni:
 - Modulo e segno
 - Complemento a **2**
 - Eccesso **2ⁿ**

000
001
010
011
100
101
110
111

Rappresentazione in modulo e segno

- Analoga a quella che usiamo nella nostra notazione decimale
- Viene dedicato un bit per il segno e **n-1** bit per il modulo
- Convenzionalmente
 - **0** rappresenta il segno **+**
 - **1** rappresenta il segno **-**
- Intervallo di numeri rappresentati con n bit

$$[-2^{n-1}+1, +2^{n-1}-1]$$

ESEMPIO

n=4 bit

intervallo [-7,+7]

5 = 0101

-5 = 1101

0000

1000

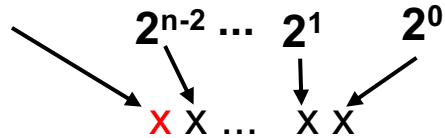
Intervallo simmetrico e doppia rappresentazione dello zero

La prima cifra non ha un significato posizionale

Rappresentazione in complemento a 2

- L'idea è di dare alla cifra più significativa il una peso negativo: -2^{n-1}

00001



Rappresentazione in complemento a 2

- Intervallo di numeri rappresentati con n bit

$[-2^{n-1}, +2^{n-1}-1] \rightarrow$ intervallo di riferimento

ESEMPIO

n=4 bit intervallo $[-8,+7]$

5 = **(0101)**_{CP2} -5 = **(1011)**_{CP2}

Intervallo asimmetrico e singola rappresentazione dello zero 0000

Complemento a 2 (continua)

- CP2 è una *notazione posizionale* Pesi: -2^{n-1} 2^{n-2} ... 2^1 2^0
- Il bit di ordine più alto (a sinistra) ha come peso una potenza di 2 negativa: -2^{n-1}
- Per i numeri negativi il primo bit è 1

ESEMPIO

$$-5 = 1011$$

$$-5 = 1 \cdot (-8) + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$

Esempi di Complemento a 2

ESEMPIO numero positivo $(6)_{10}$

calcolo il modulo **(110)** e aggiungo uno zero

$$(6)_{10} = (0110)_{CP2} = 0 \cdot (-8) + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

ESEMPIO numero negativo $(-6)_{10}$

calcolo il modulo **(110)₂** aggiungo uno zero e complemento a 2

Partendo da destra si lasciano invariati tutti i bit fino al primo 1 compreso, e poi si complementa bit a bit

0110

1010

$$(1010)_{CP2} = 1 \cdot (-8) + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

CP2 con un numero arbitrario di bit (> minimo)

$(101)_{\text{CP2}}$?

$$-4 + 1 = (-3)_{10}$$

$(1101)_{\text{CP2}}$?

$$-8 + 4 + 1 = (-3)_{10}$$

$(111111111111101)_{\text{CP2}}$?

$$(-3)_{10}$$

$$010 \rightarrow \mathbf{00000010} \quad 10101 \rightarrow \mathbf{1111110101}$$

si replica la cifra + significativa

Esercizi

- Calcolare $(15)_{10}$ in CP2 con 8 bit

modulo = 1111

01111 minimo numero di bit

$(00001111)_{CP2}$ 8 bit

- Calcolare -18 in CP2 con 8 bit

•modulo = $16+2=$ 10010

•aggiungo uno 0 010010 e complemento a 2

• 101110

• $(11101110)_{CP2}$

Rappresentazione in eccesso 2^{n-1}

- I numeri vengono rappresentati come somma fra il numero dato e una potenza di 2
- Con n bit si rappresenta l'eccesso 2^{n-1}
- Intervallo come CP2: $[-2^{n-1}, +2^{n-1}-1]$ → **intervallo di riferimento**
- Regola pratica:

I numerali in eccesso 2^{n-1} si ottengono da quelli in CP2 complementando il bit più significativo

ESEMPIO

n=4 bit: eccesso 8, intervallo $[-8, +7]$

- 3 - 3+8=5 : **0101**

+4 +4+8=12 : **1100**

A parità di bit
stesso intervallo
di CP2

Intervallo asimmetrico e singola rappresentazione dello zero 1000

4 bit CP2 vs 4 bit Ex 8 (2^{n-1})

- $(3)_{10} \rightarrow (0011)_{\text{CP2}} \quad (3+8)_{10} \quad (1011)_{\text{Ex 8}}$
- $(-3)_{10} \rightarrow (1101)_{\text{CP2}} \quad (-3+8)_{10} \quad (0101)_{\text{Ex 8}}$

Rappresentazioni in eccesso

- La rappresentazione in eccesso può essere fatta usando un numero **k** qualsiasi
- L'eccesso con una potenza di **2** è solo un caso particolare, anche se molto interessante
- Rappresentando un intero **m** in eccesso **k** con **n** bit, si rappresenta in realtà il numero positivo **k+m**
- L'intervallo rappresentabile dipende sia da **k** che da **n**:

$$[-k, 2^n - k - 1]$$

ESEMPIO

n=8, k=127 [-127,+128]

n=8, k=100 [-100,+155]

n=4, k=-17 [+17,+32]

$$k=2^{n-1}-1$$

è usato nello standard IEEE per la virgola mobile (numeri reali)

Rappresentazioni a confronto $n = 4$ Ex 2^{4-1}

Decimale	M&S	CP2	Ex 8
+ 7	0111	0111	1111
+ 6	0110	0110	1110
+ 5	0101	0101	1101
+ 4	0100	0100	1100
+ 3	0011	0011	1011
+ 2	0010	0010	1010
+ 1	0001	0001	1001
+ 0	0000	0000	1000
- 0	1000	-----	-----
- 1	1001	1111	0111
- 2	1010	1110	0110
- 3	1011	1101	0101
- 4	1100	1100	0100
- 5	1101	1011	0011
- 6	1110	1010	0010
- 7	1111	1001	0001
- 8	----	1000	0000

Numerali e numeri

- Un *numerale* è solo una stringa di cifre
- Un numerale rappresenta un numero solo se si specifica un sistema di numerazione
- Lo stesso numerale rappresenta diversi numeri in diverse notazioni

ESEMPIO

La stringa **110100** rappresenta:

- *Centodiecimilacent* in base 10
- $(+52)_{10}$ in binario naturale
- $(-12)_{10}$ in complemento a 2
- $(+20)_{10}$ in eccesso 32
- In esadecimale un numero dell'ordine di vari milioni

Addizioni binarie

- Le addizioni fra numerali binari si effettuano cifra a cifra (come in decimale) portando il riporto alla cifra successiva

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ con il riporto di } 1$$

ESEMPIO: $3 + 2 = 5$

$$\begin{array}{r} 0011 + \\ 0010 = \\ \hline 0101 \end{array}$$

Se il numero di cifre non permette di rappresentare il risultato si ha un trabocco nella propagazione del riporto

Addizioni in complemento a 2 [-8,+7]

- In CP2 somme e sottrazioni tra numerali sono gestite nello stesso modo, *ma si deve ignorare il trabocco*:

$$\begin{array}{r} 4 + \\ 2 = \\ \hline 6 \end{array} \quad \begin{array}{r} 0100 + \\ 0010 = \\ \hline 0110 \end{array}$$

- Se gli operandi hanno segno diverso il risultato è **sempre** corretto:

$$\begin{array}{r} 4 + \\ -1 = \\ \hline 3 \end{array} \quad \begin{array}{r} 0100 + \\ 1111 = \\ \hline 10011 \end{array} \quad \begin{array}{r} 1111 + \\ 1110 = \\ \hline 11101 \end{array} \quad \begin{array}{r} -1 \\ -2 \\ -3 \end{array}$$

- Se i due operandi hanno lo stesso segno e il risultato segno diverso il risultato è sbagliato

$$\begin{array}{r} 6 + \\ 3 = \\ \hline 9 \end{array} \quad \begin{array}{r} 0110 + \\ 0011 = \\ \hline 1001 \end{array}$$

(9 non cade nell'intervallo)

```
int main()
{
    int i;
    i=8000000000+8000000000;
    printf("i vale : %d",i);
    return 0;
}
```

```
1 int main()
2 {
3     int i;
4     i=8000000000+8000000000;
5     printf("i vale : %d",i);
6     return 0;
7 }
Output main.c
i vale : -1179869184
```

Numero minimo di bit (Esempio 1: $(160)_{10} \rightarrow \text{CP2}$)

- Qual è il numero minimo di bit necessari per rappresentare un numero?
- $(160)_{10}$
- $[-8, +7]$ 4
- $[-16, +15]$ 5
- ...
- $[-128, +127]$ 8
- $[-256, +255]$ 9
- $[-512, +511]$ 10

Numero di bit (Esempio 1: 160 → CP2)

- Dato il numero **m** determinare il numero minimo di cifre **n_min** necessarie per rappresentarlo in CP2 ed Eccesso 2^{n-1}
- Se $m > 0$ determinare la prima potenza $2^{n-1} > m$;
- se $m < 0$ determinare la prima potenza $2^{n-1} \geq |m|$. **m** appartiene a $[-2^{n-1}, +2^{n-1}-1]$

ESEMPIO: convertire $(+160)_{10}$ in CP2 **m positivo** → $2^{n-1} > m$

$$256 > m \rightarrow 2^8$$

intervallo con 9 bit: $[-256, +255]$

pertanto **n_min = 9**

$$160 = 128 + 32$$

$(160)_{10}$ in CP2 con 9 bit è:

-256	128	64	32	16	8	4	2	1
0	1	0	1	0	0	0	0	0

Numero di bit (Esempio 2: $256 \rightarrow \text{CP2}$)

- Dato il numero m determinare il numero minimo di cifre n_{min} necessarie per rappresentarlo in CP2 ed Eccesso 2^{n-1}
- Se $m > 0$ determinare la prima potenza $2^{n-1} > m$;
- se $m < 0$ determinare la prima potenza $2^{n-1} \geq |m|$. m appartiene a $[-2^{n-1}, +2^{n-1}-1]$

ESEMPIO: convertire $(+256)_{10}$ in CP2 m positivo $\rightarrow 2^{n-1} > m$

$$512 > m \rightarrow 2^9$$

intervallo con 10 bit: $[-512, +511]$

pertanto $n_{\text{min}} = 10$

$$256 = 256 + 0$$

$(256)_{10}$ in CP2 con 10 bit è:

-512	+256	128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	0	0	0

Numero di bit (Esempio 2: -347 → CP2)

- Dato il numero **m** determinare il numero minimo di cifre **n_min** necessarie per rappresentarlo in CP2 ed Eccesso 2^{n-1}
- Se $m > 0$ determinare la prima potenza $2^{n-1} > m$;
- se $m < 0$ determinare la prima potenza $2^{n-1} \geq |m|$. **m** appartiene a $[-2^{n-1}, +2^{n-1}-1]$

ESEMPIO: convertire $(-347)_{10}$ in CP2 **m negativo** → $2^{n-1} \geq |m|$

$$512 > |m| \rightarrow 2^9$$

intervallo con 10 bit: $[-512, +511]$

pertanto **n_min = 10**

Numero di bit (Esempio 3 -347 CP2)

- Se il numero è *negativo*:
 - determinare il numero di bit **n** → **10**
 - convertire il positivo corrispondente in notazione a **n** bit
 - complementare il numerale così ottenuto

$(347)_{10}$ in notazione a 10 bit è:

512	256	128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0	1	1

quindi, complementando a 2

- 512	256	128	64	32	16	8	4	2	1
1	0	1	0	1	0	0	1	0	1

347	1	-	significativa
173	1		
86	0		
43	1		
21	1		
10	0		
5	1		
2	0		
1	1	+	significativa
0			

Numero di bit (Esempio 4: -347 → Eccesso)

- Dato un numero **m** determinare il numero minimo di cifre **n** come abbiamo fatto per il CP2
- Aggiungere 2^{n-1} a **m** e rappresentare il risultato in binario puro su n bit

ESEMPIO: convertire $(-347)_{10}$ in eccesso 2^{n-1}

$$512 \geq 347 = 2^9$$

intervallo con 10 bit: $[-512, +511]$

pertanto $n_{min}=10$

$$-347+512 = 165$$

$$165 = 128+32+4+1$$

$(-347)_{10}$ in eccesso 2^9 è:

512	256	128	64	32	16	8	4	2	1
0	0	1	0	1	0	0	1	0	1

Numero di bit (Esempio 4: -347 → Eccesso)

- Oppure
- Prendere la rappresentazione in CP2 e complementare il primo bit
- $(1010100101)_{CP2} \leftrightarrow (0010100101)_{Ex}$
- N.B. Questo passaggio è valido SOLO se l'eccesso è una potenza esatta di 2 !!
- Negli esercizi useremo SEMPRE potenze di 2
- In eccesso i numeri negativi cominciano per 0
- Perché? Intuizione:
 - $(3)_{10}$ su 4 bit in binario puro o in CP2 vale $(0011)_{CP2}$
 - aggiungere 8 $1000 \rightarrow (1011)_{Ex}$
 - $(-3)_{10}$ in CP2 vale $(1101)_{CP2}$
 - aggiungere 8 $-3+8=5 \rightarrow (0101)_{Ex}$

Rappresentazione parte frazionaria in base 2

- $(14.65)_{10}$ 0.65 ?
- si decide quanti bit dedicare alla parte frazionaria es. 4 bit
- si moltiplica la parte frazionaria 4 volte per due prendendo la parte intera e sottraendo se il numero ottenuto è maggiore di 1...

0.65×2	$= 1.3$	1 Cifra più significativa;	$1.3 - 1 =$
0.3×2	$= 0.6$	0	
0.6×2	$= 1.2$	1	$1.2 - 1 =$
0.2×2	$= 0.4$	0 Cifra meno significativa	

$(1110.1010)_2$

$1 \ 1 \ 1 \ 0 \ . \ 1 \ 0 \ 1 \ 0 \ \rightarrow 14 + 0.5 + 0.125 = 14.625 \sim 14.65$ approssimazione!
 $2^3 \ 2^2 \ 2^1 \ 2^0 \ . \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4}$

N.B. In un calcolatore i reali vengono rappresentati in modo differente: virgola mobile!

Errori nella rappresentazione della parte frazionaria in base 2

- 0.65 ?
- ma quante bit servono per rappresentare correttamente 0.65?

```
1) 0.65*2      = 1.3      1      1.3 - 1=
2) 0.3*2       = 0.6      0
3) 0.6*2       = 1.2      1      1.2 - 1=
4) 0.2*2       = 0.4      0
5) 0.4*2       = 0.8      0
6) 0.8*2       = 1.6      1      1.6 - 1= 0.6
7) 0.6*2       vai al 3)
```

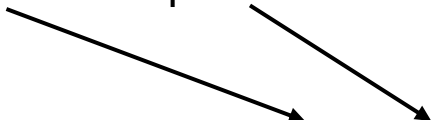
$$(0.65)_{10} = (0.101001\overline{)}_2$$

- infiniti!!! rappresentando 0.65 in base 2 si commette un errore...

```
In [1]: 0.65-0.5
Out[1]: 0.150000000000000002
```

- mentre $(1/3)_{10} = (0.1)_3 \dots$

Rappresentazione in virgola mobile in base 2

- E' opportuno gestire rappresentazioni <mantissa> <esponente> dove la mantissa è contenuta in $[2^0, 2^1) = [1, 2)$ 
- mantissa $[base^0, base^1)$ in base 10 $[10^0, 10^1) = [1, 10)$ e.g., $9.32 \cdot 10^7$
- Esempio $(14)_{10} \rightarrow 1110$ ovvero 1110.0 ovvero $1.110 \cdot 2^3$
- $(14.65)_{10} \sim (1110.1010)$ ovvero $1.1101010 \cdot 2^3$

Esercizio 1

- dato il numero $(-74)_{10}$ calcolare la sua rappresentazione in
- CP2
- Ex 2^9
- usando 10 bit

Esercizio 1

- modulo di $(-74)_{10} \rightarrow 1001010$ ($74=64+8+2$)
- CP2
 - aggiungo uno zero davanti **01001010**
 - ricopio da destra fino al primo 1 compreso poi complemento **10110110**
 - estendo il bit più significativo raggiungendo i 10 bit (**1110110110**)_{CP2}
- Ex 2^9
 - complemento il primo bit di CP2 \rightarrow (**0110110110**)_{Ex}
 - oppure (tutto in base 10) $-74+2^9=-74+512=438 \rightarrow (0110110110)_{Ex}$

Esercizio 1

- modulo di $(-74)_{10} \rightarrow 1001010$ ($74=64+8+2$)
- CP2
 - aggiungo uno zero davanti **01001010**
 - ricopio da destra fino al primo 1 compreso poi complemento **10110110**
 - estendo il bit più significativo raggiungendo i 10 bit (**1110110110**)_{CP2}
- Ex 2^9
 - complemento il primo bit di CP2 \rightarrow (**0110110110**)_{Ex}
 - oppure (tutto in base 10) $-74+2^9=-74+512=438 \rightarrow (0110110110)_{Ex}$

Esercizio 2

- dato il **numerale** a 16 bit in esadecimale FFA5
- a) indicare il numerale in binario e calcolare l'ordine di grandezza in base 2 e base 10 interpretandolo in
 - CP2
 - Ex
 - binario puro
- b) rappresentare con parte frazionaria della mantissa limitata a 8 bit il valore espresso in binario puro

Esercizio 2

- FFA5 → Numerale!
- CP21111 1111 1010 0101
 - è un numero negativo
 - rimuovo i bit più significativi irrilevanti ~~1111 1111~~ 1010 0101
 - ricopio da destra sino al primo 1 e complemento 0101 101¹
 - 0101 1011 = 1.01101 $2^6 \rightarrow$ ordine di grandezza $2^6 \sim 10^{1.8}$ (divido per 10 e moltiplico per 3)
- Ex $2^{15} = 2^{10} 2^5$
 - è un numero positivo (1111 1111 1010 0101)_{Ex}
 - lo trasformo in CP2 complementando il primo bit (0111111110100101)_{CP2}
 - 0111111110100101 = 1.11111110100101 2^{14} ovvero $10^{4.2}$

Esercizio 2

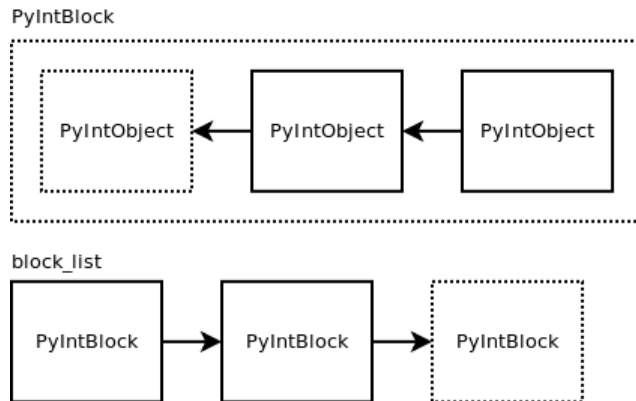
- 1111 1111 1010 0101
- binario puro
 - $1.111111110100101 \cdot 2^{15}$
 - ordine di grandezza $2^{15} \sim 10^{4.5}$ (divido per 10 e moltiplico per 3)
- se ho solo 8 bit per la mantissa
 - $1.1111111 \cdot 2^{15}$
- Idea! Ma, dato che la mantissa comincia sempre per 1. perché non rappresentare solo la parte frazionaria? Guadagniamo una cifra...
- parte frazionaria mantissa = 11111111 esponente = 15
- e se il numero è negativo? un bit per il segno!

Rappresentazione Interi in Python

- Python usa una rappresentazione degli interi a precisione arbitraria. In pratica usa un numero arbitrario di locazioni di memoria da 32 bit ciascuna per rappresentare un intero

Ogni blocco da 32 bit contiene un numero binario senza segno da 30 bit, il segno è contenuto a parte in un bit specifico.

Rappresentazione in modulo e segno



Rappresentazione Interi in Python (base 2^{30})!

La rappresentazione di $n=123456789101112131415$ avviene rappresentando n in base $b=2^{30}$. Non esistono b simboli e ogni cifra è rappresentata da un numero binario a 30 bit

Per rappresentare n occorrono 3 cifre: **abc** il cui peso è $(2^{30})^2$ $(2^{30})^1$ $(2^{30})^0$

a=107 b=87719511 c=437976919

$$107 * (2^{30})^2 + 87719511 * (2^{30})^1 + 437976919 (2^{30})^0 = 123456789101112131415$$

```
In [18]: b=2**30
```

```
In [19]: 107*b**2 + 87719511*b + 437976919
```

```
Out[19]: 123456789101112131415
```

In memoria abbiamo una struttura in cui le tre cifre sono indicate in ordine inverso (dalla meno significativa a quella più significativa)

ob_size	3		
ob_digit	(437976919)	(87719511)	(107)
	30 bit	30 bit	30 bit

Rappresentazione Interi in Python

- Le operazioni sono realizzate spezzandole in operazioni su numeri di 30 bit (con riporto tra un blocco e un altro)
- Ad esempio, per sommare due numeri si sommano prima i due blocchi meno significativi, si calcola il risultato e il riporto e poi si passa ai secondi due blocchi (aggiungendo il riporto).
- Altre operazioni anche più complesse vengono sempre ridotte a sequenze di operazioni su blocchi da 30 bit
- In qualche versione è comparso un numero intero massimo **sys.maxint** per problemi legati alla gestione delle stringhe ma ora è scomparso. Al suo posto **sys.maxsize=9223372036854775807** 'can be used as an integer larger than any practical list or string index'
- se vi serve un numero più grande di qualunque numero intero rappresentabile in python usate **float('inf')** 😊