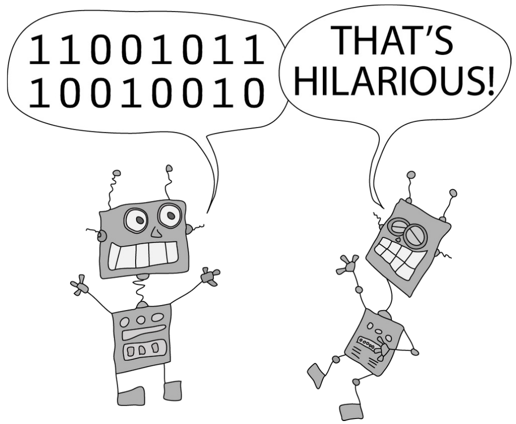
UNIVERSITÀ DI PARMA

sistema binario



analogico e digitale

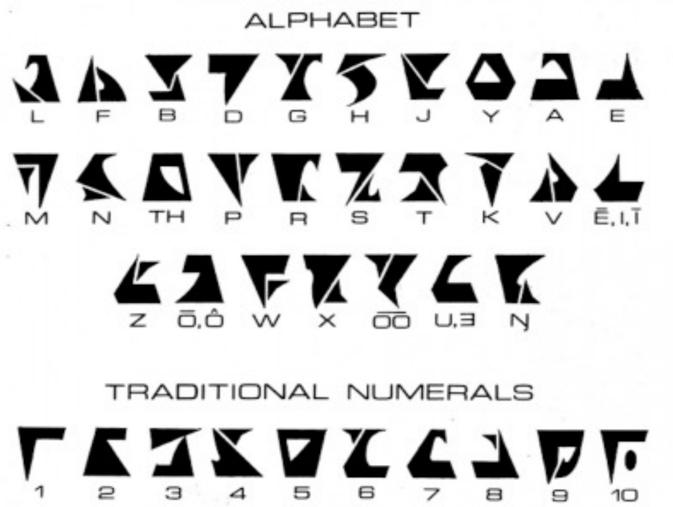
- una ***grandezza*** (*fisica o astratta*) può essere rappresentata in due forme
  - ***analogica***
    - insieme di valori ***continuo***
  - ***digitale***
    - insieme di valori ***discreto*** (tutti i punti sono isolati)

approssimazione discreta

- alcune informazioni sono *intrinsecamente discrete*
  - informazioni “artificiali”, es. un testo scritto
  - scala atomica o subatomica …
- molte grandezze fisiche hanno forma *continua*
  - per gestirle con i computer è necessaria una ***rappresentazione digitale***
  - ***approssimazione*** del valore analogico
  - ***errore*** dipende dalla precisione della rappresentazione digitale scelta

codice

- sistema basato su ***simboli***, che permette la ***rappresentazione*** dell'informazione
  - ***simbolo***: elemento atomico
  - ***alfabeto***: insieme dei simboli possibili (A)
  - ***cardinalità*** del codice: numero di simboli dell'alfabeto
  - ***stringa***: sequenza di simboli (s ∈ A*)
  - ***linguaggio***: insieme *stringhe ben formate* (L ⊆ A*)



codice posizionale

- un numero naturale può essere rappresentato con una ***notazione posizionale***
  - $N = c_0 \cdot base^0 + c_1 \cdot base^1 + \ldots + c_n \cdot base^n$
  - es. $587_{10} = 7 \cdot 10^0 + 8 \cdot 10^1 + 5 \cdot 10^2$
- sistemi di numerazione posizionali di uso comune
  - ***decimale*** (base 10; c: 0-9)
  - ***binario*** (base 2; c: 0-1)
  - ***esadecimale*** (base 16; c: 0-9, A-F)

Hexadecimal Numbering

Decimal and Binary equivalents of 0 to F Hexadecimal

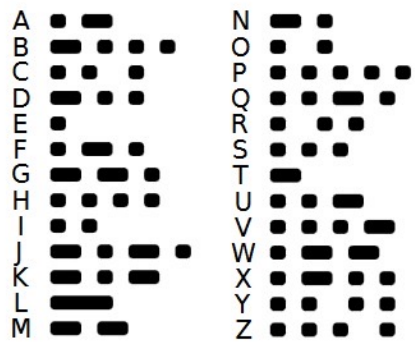
Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Selected Decimal, Binary and Hexadecimal equivalents

Decimal	Binary	Hexadecimal
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
3	0000 0011	03
4	0000 0100	04
5	0000 0101	05
6	0000 0110	06
7	0000 0111	07
8	0000 1000	08
10	0000 1010	0A
15	0000 1111	0F
16	0001 0000	10
32	0010 0000	20
64	0100 0000	40
128	1000 0000	80
192	1100 0000	C0
202	1100 1010	CA
240	1111 0000	F0
255	1111 1111	FF

codifica dell'informazione

- ***codifica***: regole di ***corrispondenza*** per passare da un certo codice ad un altro
- corrispondenza ***biunivoca***
  - tra una stringa di un codice
  - e una stringa di un altro codice
- ad una certa stringa in un alfabeto ricco di simboli, corrisponde una stringa più lunga in un alfabeto più ridotto
- ingegneria = .. -. --. . --. -. . . .-. .. .-

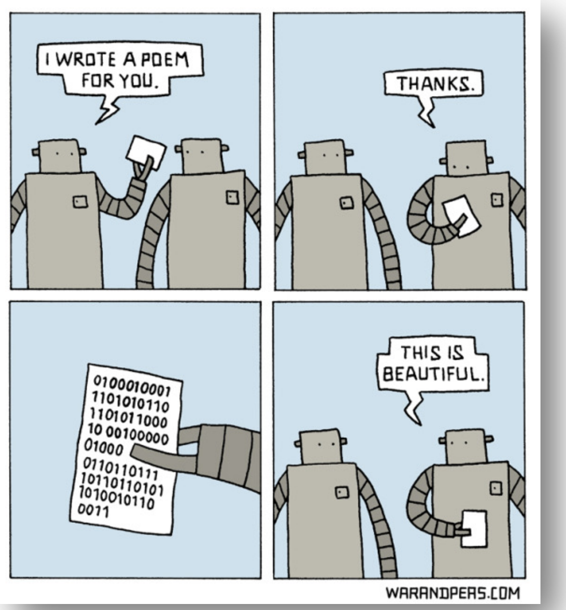


sistema binario

numeri binari

codice binario

- ***base 2***
- alfabeto (0,1)
- informazione digitale nei calcolatori rappresentata con una sequenza di 0 e 1
  - sistema ideato da *Leibniz*, ~1700
  - primo calcolatore binario ***Zuse***, ~1940
- ***bit*** = elemento di una sequenza
- ***byte*** = sequenza di 8 bit (***nibble*** 4 bit)
- ***MSB*** (*Most significant bit*): bit più a sinistra nella sequenza
- ***LSB*** (*Least significant bit*): bit più a destra nella sequenza



codifica decimale → binaria

- (1) dividere il numero decimale per 2
- (2) il resto è il valore del nuovo bit, a sinistra (loop)
- continuare a dividere per 2 il quoziente finché non si annulla
- Es.: $35_{10} = 100011_2$

n	n // B	n % B	peso
35	17	1	$1 = 2^0$
17	8	1	$2 = 2^1$
8	4	0	$4 = 2^2$
4	2	0	$8 = 2^3$
2	1	0	$16 = 2^4$
1	0	1	$32 = 2^5$

UNIVERSITÀ DI PARMA

numeri naturali

- rappresentare un numero ***naturale*** N in forma ***binaria***
  - occorrono K bit, t.c. $\mathbf{2^K > N}$
  - es. 4 bit per numeri naturali da 0 a 15
- il calcolatore assegna un ***numero fisso di bit*** per i diversi tipi di informazione
- situazioni di valori non rappresentabili
  - ***overflow***, ***underflow***

potenze di 2

- proprietà delle potenze

$$a^x \cdot a^y = a^{x+y}, a^x/a^y = a^{x-y}, (a^x)^y = (a^y)^x = a^{x \cdot y}$$

- $1K = 2^{10} \simeq 10^3$
- $1M = 2^{20} \simeq 10^6$ (*mega*)
- $1G = 2^{30} \simeq 10^9$ (*giga*)

$2^1 = 2$	$2^9 = 512$
$2^2 = 4$	$2^{10} = 1024$
$2^3 = 8$	$2^{11} = 2048$
$2^4 = 16$	$2^{12} = 4096$
$2^5 = 32$	$2^{13} = 8192$
$2^6 = 64$	$2^{14} = 16384$
$2^7 = 128$	$2^{15} = 32768$
$2^8 = 256$	$2^{16} = 65536$

esadecimale (hex)

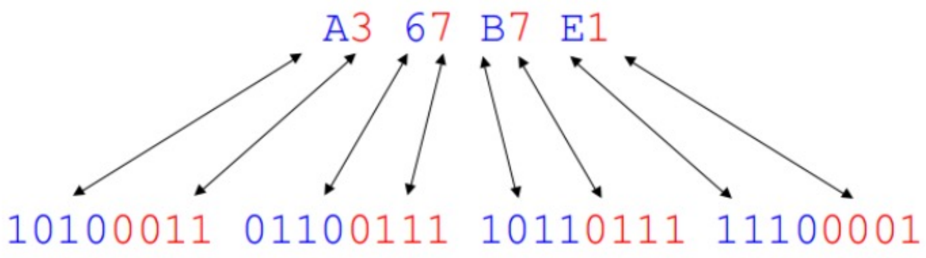
Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex
00	0000 0000	00	16	0001 0000	10	32	0010 0000	20
01	0000 0001	01	17	0001 0001	11	33	0010 0001	21
02	0000 0010	02	18	0001 0010	12	34	0010 0010	22
03	0000 0011	03	19	0001 0011	13	35	0010 0011	23
04	0000 0100	04	20	0001 0100	14	36	0010 0100	24
05	0000 0101	05	21	0001 0101	15	37	0010 0101	25
06	0000 0110	06	22	0001 0110	16	38	0010 0110	26
07	0000 0111	07	23	0001 0111	17	39	0010 0111	27
08	0000 1000	08	24	0001 1000	18	40	0010 1000	28
09	0000 1001	09	25	0001 1001	19	41	0010 1001	29
10	0000 1010	0A	26	0001 1010	1A	42	0010 1010	2A
11	0000 1011	0B	27	0001 1011	1B	43	0010 1011	2B
12	0000 1100	0C	28	0001 1100	1C	44	0010 1100	2C
13	0000 1101	0D	29	0001 1101	1D	45	0010 1101	2D
14	0000 1110	0E	30	0001 1110	1E	46	0010 1110	2E
15	0000 1111	0F	31	0001 1111	1F	47	0010 1111	2F

Bin ↔ Hex

ogni gruppo di 4 bit

- 16 configurazioni diverse ($2^4 = 16$)

corrisponde ad uno dei 16 simboli esadecimali



somma e sottrazione binaria

```
BINARY
    1   1
0 0 0 1 0 1 1 0 +
0 0 0 1 0 1 0 1 =
-----------------
0 0 1 0 1 0 1 1
```

```
BINARY
            0 10
0 0 0 0 1 1 1 0 -
0 0 0 0 0 1 0 1 =
-----------------
0 0 0 0 1 0 0 1
```

moltiplicazione binaria

BINARY

```
      1 0 1 1 x
      1 1 0 1 =
      ---------
      1 0 1 1 +
    0 0 0 0
  -------------
    0 1 0 1 1 +
  1 0 1 1
 ---------------
  1 1 0 1 1 1 +
1 0 1 1
----------------
1 0 0 0 1 1 1 1
```

divisione binaria

```
BINARY
1 0 1 1 0 1 : 1 1
0 0          ---------
-----         0 1 1 1 1
1 0 1 -
  1 1
-------
  1 0 1 -
    1 1
  -------
    1 0 0 -
      1 1
    -------
        1 1 -
        1 1
        -----
        0 0
```

numeri interi

- rappresentazione ***binaria*** dei numeri ***interi***
- occorre rappresentare anche i numeri ***negativi***
  - necessario riservare un ***bit*** per il ***segno***
  - $\Rightarrow$ si ***dimezza*** il massimo modulo ammesso
- rappresentazione modulo e segno
  - il ***primo*** bit indica il ***segno***
  - ***0 positivo, 1 negativo***

Binary	Unsi-gned	Signed magni-tude	Two's comple-ment
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	-0	-8
1001	9	-1	-7
1010	10	-2	-6
1011	11	-3	-5
1100	12	-4	-4
1101	13	-5	-3
1110	14	-6	-2
1111	15	-7	-1

complemento a 2

- rappresentazione ***alternativa***
  - ***diversa da modulo e segno***!
- numero negativo, ottenuto dal suo opposto positivo
  - complemento il numero
    - gli 1 diventano 0 e viceversa
- sommo 1
  - anche così, il primo bit indica il segno
  - 0 positivo, 1 negativo
- ***attenzione***: bisogna conoscere ***codifica*** e ***numero bit***
  - esempi seguenti: ogni intero con segno
    è memorizzato in un singolo byte

Binary	Unsi-gned	Signed magni-tude	Two's comple-ment
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	-0	-8
1001	9	-1	-7
1010	10	-2	-6
1011	11	-3	-5
1100	12	-4	-4
1101	13	-5	-3
1110	14	-6	-2
1111	15	-7	-1

complemento a 2: esempio

- rappresentazione con un byte, **+35** è in binario: **00100011**
- numero –35, in ***modulo e segno***: **10100011**
- Numero –35, in ***complemento a due***: **11011101**

```
0 0 1 0 0 0 1 1 ¬
-----------------
1 1 0 1 1 1 0 0 +
              1 =
-----------------
1 1 0 1 1 1 0 1
```

¬: *complemento semplice, bit a bit*

somma con segno

- sommare **12** e **-35** su ***8 bit***, ***modulo e segno***
  - sottrazione tra 35 e 12
  - cambio di segno
- stessa operazione, ***complemento a due***
  - semplice somma: 12 + -35 = -23

```
0 0 0 0 1 1 0 0 +
1 1 0 1 1 1 0 1 =
----------------
1 1 1 0 1 0 0 1
```

numeri reali

- ***insieme continuo***, per grandezze analogiche
  - rappresentabili solo in modo ***approssimato*** (***numeri razionali***)
- parte ***frazionaria***
  - $F = c_{-1} \cdot base^{-1} + \dots + c_{-n} \cdot base^{-n}$
  - $0.365_{10} = 3 \cdot 10^{-1} + 6 \cdot 10^{-2} + 5 \cdot 10^{-3}$
  - $0.1011_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}$
- due rappresentazioni alternative
  - ***virgola fissa***: ***segno, parte intera, parte decimale***
  - ***virgola mobile***: ***segno, mantissa, esponente***

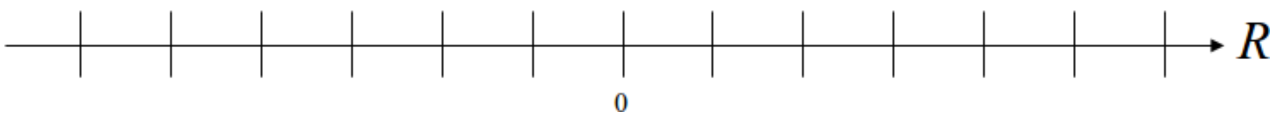
parte frazionaria in binario

- ***(1) moltiplicare*** la parte frazionaria per 2
- ***(2) assegnare*** la parte intera del risultato come valore del ***bit*** (***loop***)
  - continuare a moltiplicare per 2 la parte frazionaria del risultato...
  - finché non si annulla

fract	fract*B	int	peso
0,375	0,750	0	2^{-1}
0,750	1,500	1	2^{-2}
0,500	1,000	1	2^{-3}

virgola fissa

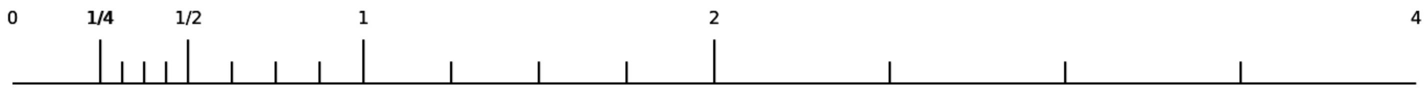
- numero espresso come: **`r = (i, f)`**
  - ***i*** è la ***parte intera***, ***n_1 bit***
  - ***f*** è la ***parte frazionaria***, ***n_2 bit***
- precisione costante lungo l'asse reale
  - es. f di 3 bit, valori consecutivi sempre distanziati di **`1/8`**
    - tra ciascun intero e il successivo, possiamo rappresentare ***8 valori***



UNIVERSITÀ DI PARMA

virgola mobile

- virgola mobile (***floating point***)
- numero espresso come: $\mathbf{r = \pm(1+f) \cdot 2^e}$
  - **e** è l'esponente intero (o caratteristica), $\mathbf{n_1}$ bit
  - **f** è la parte frazionaria ($0 \leq f < 1$), $\mathbf{n_2}$ bit
  - ***2*** è la ***base***, ***1+f*** è anche detto ***mantissa***
- precisione variabile lungo l'asse reale
  - $f \in \{0, 1/4, 2/4, 3/4\}$, 2 bit
  - $e \in \{-2, -1, 0, 1\}$, 2 bit

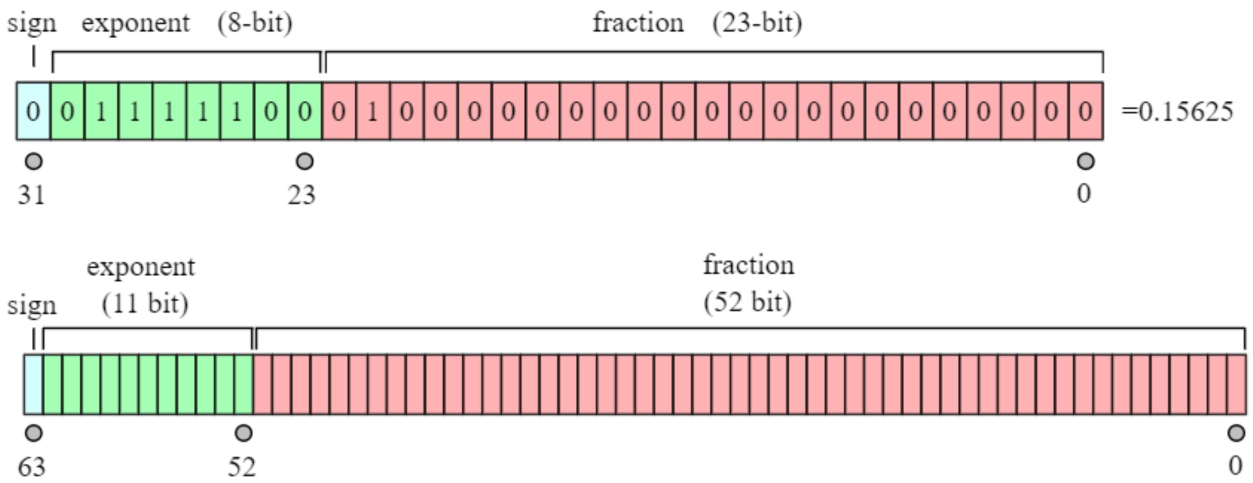


http://www.mathworks.com/company/newsletters/news_notes/pdf/Fall96Cleve.pdf

IEEE 754 single & double

Institute of Electrical and Electronic Engineers

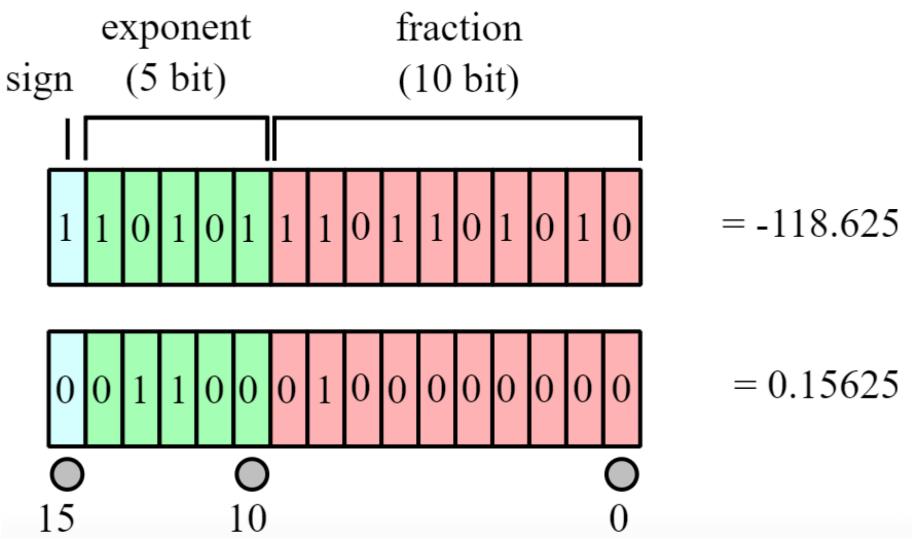
- precisione ***singola***: ***32 bit***
  - 1 x segno, 8 x esponente, 23 x frazione
- precisione ***doppia***: ***64 bit***
  - 1 x segno, 11 x esponente, 52 x frazione



IEEE 754 half-precision

- rappresentazione usata nelle GPU, per velocizzare i calcoli
  - $-118.625 = -1110110.101_2 = -1.110110101_2 \times 2^6$
  - all'esponente, su 5 bit, bisogna sommare 15 ($=2^{(5-1)} - 1$)

Graphics **P**rocessing **U**nit processore grafico tipo particolare di coprocessore



uguaglianza e prossimità

- approssimazione discreta dei numeri reali
- ***attenzione ai confronti di uguaglianza***

```
>>> 0.2 + 0.1 == 0.3
False
```

```
for i in range(360):
   a = math.radians(i) print(math.sin(a) ** 2 + math.cos(a) ** 2 == 1)
# fails ~¼ of times
```

```
>>> abs(x - y) <= 10 ** -9
True
>>> math.isclose(x, y)
True
```

https://www.codecademy.com/resources/docs/python/math-module/math-isclose