Architettura dei Calcolatori 1

Lezione 2 | 26 Febbraio

Università degli Studi di Modena e Reggio Emilia — Unimore

1. Rappresentazione binaria dell'informazione

Tutta l'informazione all'interno di un computer è codificata con sequenze di **0** e **1**. Perché due valori? È la mappatura più semplice su un segnale fisico discreto (transistor ON/OFF, condensatore carico/scarico).

Unità	Definizione
Bit	L'unità minima di informazione: 0 oppure 1. (Binary Digit.)
Byte	8 bit — unità standard di misura della memoria.
Word	Sequenza di bit nativa dell'architettura del processore (vedi tabella).

Dimensione	Esempi	Periodo
8 bit (1 B)	Intel 8080, Zilog Z80	1975–1994
16 bit (2 B)	Intel 8086, Motorola 68000	1979–1994
32 bit (4 B)	IA-32, ARMv3–v7, RISC-V	1993–2010
64 bit (8 B)	AMD64, IA-64, ARMv8, PowerPC	2006–oggi

Nota

Più grande è la word, più informazione può essere rappresentata e processata *in un singolo ciclo di clock.*

2. Sistema decimale posizionale — ripasso

Un numero intero A in base 10:

$$A = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \cdots + a_1 \cdot 10^1 + a_0 \cdot 10^0$$

Esteso alla parte frazionaria: i pesi continuano con potenze negative.

Proprietà fondamentali:

1. Massimo numero rappresentabile con n cifre: $\underbrace{99\cdots9}_{n} = 10^n - 1$
2. Per rappresentare K configurazioni servono almeno x cifre, dove 10^x è la più piccola potenza di 10 maggiore di K.

Le stesse identiche proprietà valgono in qualunque altra base — cambia solo la base.

3. Notazione posizionale in base 2

Cifre disponibili: $\{0, 1\}$. Formula:

$$A = a_N \cdot 2^N + \cdots + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + \cdots$$

Binario	Calcolo	Decimale
`10`	$1 \cdot 2^1 + 0 \cdot 2^0$	2
`110`	$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$	6
`1`	$1 \cdot 2^0$	1

Attenzione

`10` si legge "uno-zero", NON "dieci". `110` si legge "uno-uno-zero", NON "centodieci".

Proprietà fondamentali in base 2:

1. Massimo con N bit: $\underbrace{11\cdots1}_{N} = 2^N - 1$
2. Per rappresentare $K = 25$ configurazioni: $2^5 = 32 > 25$ e $2^4 = 16 < 25 \Rightarrow$ servono 5 bit.

4. Conversioni

4.1. Base 10 → Base 2: divisioni successive per 2

Dividi ripetutamente per 2 finché il quoziente è 0. **I resti letti dal basso verso l'alto** danno la rappresentazione binaria.

Dividendo	Quoziente	Resto
13	6	1
6	3	0
3	1	1
1	0	1 ← stop

Leggendo i resti dal basso: $13_{10} = 1101_2$

4.2. Base 2 → Base 10: somma pesata

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13_{10} \checkmark$$

5. Tabelle di riferimento

5.1. Binario ↔ Decimale (0–18)

Dec	Bin	Dec	Bin
0	0000	10	1010
1	0001	11	1011
2	0010	12	1100
3	0011	13	1101
4	0100	14	1110
5	0101	15	1111
6	0110	16	10000
7	0111	17	10001
8	1000	18	10010
9	1001		

Nota

Bisogna familiarizzare almeno fino al 15 (4 bit = 1 nibble = mezza cifra hex).

5.2. Potenze di 2 da memorizzare

2^n	Valore	2^n	Valore
2^0	1	2^8	256
2^1	2	2^{10}	1 024 (≈1K)
2^2	4	2^{16}	65 536 (≈64K)
2^3	8	2^{20}	1 048 576 (≈1M)
2^4	16	2^{24}	≈16 milioni
2^5	32	2^{30}	≈1 miliardo (1G)
2^7	128	2^{32}	≈4 miliardi

6. Aritmetica binaria

6.1. Addizione

$$0+0=0 \qquad 0+1=1 \qquad 1+0=1 \qquad 1+1=0 \text{ (con riporto di 1)}$$

L'`1+1` è come il `9+1` in base 10: dà 0 con riporto.

```
  1101    (13)
+ 1011    (11)
------
 11000    (24)   v
```

6.2. Sottrazione

$$0-0=0 \qquad 0-1=1 \text{ (con prestito)} \qquad 1-0=1 \qquad 1-1=0$$

```
  1101    (13)
- 1011    (11)
------
  0010    (2)    v
```

6.3. Moltiplicazione e divisione per potenze di 2

Proprietà chiave: shift

- Moltiplicare per 2^n $\equiv$ **shift a sinistra** di n posizioni
- Dividere per 2^n $\equiv$ **shift a destra** di n posizioni

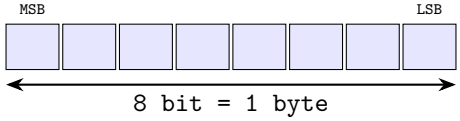
Esempio: $5 \cdot 4 = 5 \cdot 2^2 = 0101 \rightarrow 010100 = 20$ ✓

7. Rappresentazione degli interi in memoria

Tipo C	Dimensione	Range (unsigned)
`char`	1 byte (8 bit)	0 . . . 255
`short`	2 byte (16 bit)	0 . . . 65 535
`int`	4 byte (32 bit)	0 . . . 4 294 967 295
`long long`	8 byte (64 bit)	0 . . . $\approx 18 \cdot 10^{18}$

MSB e LSB

- **MSB** (Most Significant Bit): bit più a **sinistra**, peso 2^{n-1}
- **LSB** (Least Significant Bit): bit più a **destra**, peso 2^0



7.1. Il problema dell'overflow

I numeri naturali sono infiniti, ma i processor lavorano con un numero **fisso** di bit. Se il risultato supera il range rappresentabile: **overflow** — il risultato non è più corretto.

```
  11100000  (224, 8 bit)
x 2  (= shift sx di 1)
----------
 100000000  -> il 9° bit scappa fuori dagli 8 disponibili
             risultato: 00000000 (0) invece di 448  !
```

Importante

L'overflow è silenzioso: il programma non si accorge di nulla e lavora con dati sbagliati. Scegliere il tipo di dato giusto in base al range operativo è **cruciale**.

8. Rappresentazione dei numeri negativi

8.1. Tentativo 1 — Modulo e segno (e perché non funziona)

Riservo il bit MSB per il segno (0 = positivo, 1 = negativo).

Problema: l'aritmetica si rompe:

```
 +1 = 001
 -2 = 110
 -------
    111  -> interpretato come -3  X  (dovrebbe essere -1)
```

Il bit di segno non partecipa alle operazioni $\Rightarrow$ non va bene.

8.2. Complemento a 2 (Two's Complement)

Algoritmo del Complemento a 2

Per ottenere la rappresentazione di $-x$ da x:

1. Scrivi x in binario.
2. **Inverti tutti i bit** (complemento a 1).
3. **Aggiungi 1**.

Esempio: -5 in 4 bit

$$+5 = 0101 \xrightarrow{\text{NOT}} 1010 \xrightarrow{+1} 1011 = -5$$

Verifica: il bit MSB ha peso **negativo** (-2^{n-1}):

$$\texttt{1011} = -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -8 + 0 + 2 + 1 = -5 \checkmark$$

Test operativo:

```
  0001   (+1)
+ 1011   (-5)
------
  1100   -> -1*2^3 + 1*2^2 + 0 + 0 = -8+4 = -4   v (1+(-5)=-4)
```

8.2.1. Range con n bit in complemento a 2

$$\text{minimo: } \overbrace{1\underbrace{00\cdots0}_{n-1}}^{-2^{n-1}} \qquad \text{massimo: } \overbrace{0\underbrace{11\cdots1}_{n-1}}^{+2^{n-1}-1}$$

Con 3 bit come esempio completo:

Bit pattern	Unsigned	Signed (2's compl.)
000	0	0
001	1	+1
010	2	+2
011	3	+3
100	4	−4
101	5	−3
110	6	−2
111	7	−1

Il contatore "si avvolge": da `011` (+3) si aggiunge 1 e si arriva a `100` (−4).

Nota

Il numero totale di configurazioni è lo stesso dell'unsigned (2^n) — non si "perde" nulla. Si sposta semplicemente dove si mette il punto di riferimento.

8.2.2. Sign Extension

Quando si aumenta il numero di bit di una variabile, il bit MSB si **propaga** a sinistra.

−5 in 4 bit	`1011`
−5 in 8 bit	`11111011` *(gli 1 si propagano)*
+5 in 4 bit	`0101`
+5 in 8 bit	`00000101` *(gli 0 si propagano)*

9. Sistemi esadecimale e ottale

Utili perché permettono conversioni **rapidissime** da e verso il binario.

Base	Cifre	Conversione da binario
Ottale (8)	$\{0,\ldots,7\}$	Raggruppa i bit in blocchi di **3** da destra
Esadecimale (16)	$\{0,\ldots,9,A\text{–}F\}$	Raggruppa i bit in blocchi di **4** da destra

In esadecimale: $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$.

9.0.1. Esempi: 111000110101_2

→ **Ottale (blocchi da 3):**

$$\underbrace{111}_{7}\ \underbrace{000}_{0}\ \underbrace{110}_{6}\ \underbrace{101}_{5} \quad \Rightarrow \quad 7065_8$$

→ **Esadecimale (blocchi da 4):**

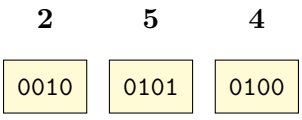
$$\underbrace{1110}_{E}\ \underbrace{0011}_{3}\ \underbrace{0101}_{5} \quad \Rightarrow \quad E35_{16}$$

Attenzione

Si aggiungono zeri a **sinistra** per completare i blocchi, non a destra (aggiungere zeri a destra sarebbe moltiplicare per la base!). Raggruppa **sempre da destra**.

10. BCD — Binary Coded Decimal

Ogni **cifra decimale** è codificata individualmente con 4 bit.



254_{10} `in BCD = 0010 0101 0100 (12 bit)`

Pro	Contro
Nessun errore di conversione decimale	Codice ridondante (spreca bit)
Precisione nei calcoli decimali	Molto più grande del binario puro
Usato nelle calcolatrici tascabili	Non efficiente per uso generale

Nota

In BCD ogni gruppo di 4 bit rappresenta una cifra da 0 a 9. Le 6 combinazioni da `1010` a `1111` (10–15) **non vengono mai usate**: codice ridondante.