2. Il computer funziona secondo una logica binaria! - Il bit e il Byte

2a - COMPUTER: MACCHINA BINARIA

Il funzionamento del computer si basa tutto sulla presenza/assenza di segnali elettrici all'interno dei milioni e milioni di circuiti che lo compongono.

L'attività del computer consiste essenzialmente nella continua rapidissima combinazione e propagazione di tutti questi segnali al proprio interno, al ritmo del "**clock**"
(un orologio interno, che "batte" 2-3 o più miliardi di volte ogni secondo),
e sotto il controllo del **microprocessore**.

Tutti i dispositivi di memoria rispecchiano un dualismo:
(presente/assente, riflettente/opaco, orario/antiorario ...);
insomma, l'elemento minimo di memoria è sempre un micro-dispositivo
che può trovarsi in UNO E UNO SOLO FRA 2 STATI FISICI BEN DISTINTI.

UN "QUALCOSA" CHE POSSA ASSUMERE SOLTANTO UNO FRA 2 POSSIBILI STATI VIENE DETTO "BIT" (da Binary digIT, ovvero "cifra binaria").

I due stati in cui può trovarsi il bit sono convenzionalmente rappresentati con "0" e "1".

memorie a semiconduttore	RAM, ROM, chiavette USB, memorie FLASH, elementi di memoria presenti nella CPU (=registri)	BIT = presenza o assenza di una debole tensione elettrica
memorie magnetiche	Hard Disk Floppy Disk (ora non più usato)	BIT = microarea orientata magneticamente in senso orario oppure antiorario
memorie ottiche	CD DVD	BIT = microarea che riflette un raggio laser oppure al contrario non lo riflette

I bit sono organizzati in gruppi di 8. **Una sequenza di 8 bit prende il nome di BYTE** (leggi *bàit*).
Noi scriveremo Byte sempre con la maiuscola, per abituarci al fatto che le abbreviazioni sono B=Byte, b=bit

Dal momento che ogni singolo bit può assumere due stati (0 oppure 1),
un Byte può assumere tutti gli stati da 00000000 a 11111111, con tutte le situazioni intermedie,
per un totale di 2^8=256 diverse combinazioni (perché 2^8? Vedi il paragrafo successivo).

Un Byte, ad esempio 01100010, potrà, a seconda del contesto, indicare:
- **un dato numerico, ad esempio il numero "98"**
- **un dato non numerico, ad es. la lettera "b minuscola", oppure il codice di un colore**
- **il codice di un'istruzione, ad es. l'istruzione di "somma", o di attivazione della stampante**
- **l'indirizzo (=il numero d'ordine) della cella di memoria dove risiede un certo dato, o istruzione.**

Allora, ricapitoliamo:
in un computer, le istruzioni di un programma, e i dati su cui il programma è chiamato a operare, sono codificati come sequenze di "0" e di "1" (bit), organizzate in gruppi di otto (Byte) e fisicamente realizzate, nella RAM (memoria di lavoro) o sulle "memorie di massa" (HD, CD, DVD, …), da dispositivi di natura diversa ma accomunati dal fatto che ciascuno di essi può, istante per istante, trovarsi in uno e uno solo fra due stati fisici ben distinti.

Il microprocessore è in grado di "MANIPOLARE" queste sequenze di 0 e di 1, seguendo le istruzioni contenute nei programmi, in modo da realizzare:
- **operazioni logiche e aritmetiche;**
- **la selezione della cella di RAM dalla quale prelevare l'istruzione che è corretto eseguire in una data fase di un processo, o dalla quale/nella quale prelevare/riversare un dato;**
- **comandi vari (es. illuminazione con un colore piuttosto che un altro di un dato puntino sullo schermo, attivazione della stampante, ecc. ecc. ecc.)**

I multipli del Byte, usati per misurare la capienza o **"capacità" delle varie memorie**, sono:
il KiloByte (KB), il MegaByte (MB), il GigaByte (GB), il TeraByte (TB).

	Fuori dall' Inform.	A volte, in Inform.	PRECISAZIONI a pag. 25	Ogni multiplo è $1024=2^{10}$ volte il precedente; la scelta di questo numero (anziché $1000=10^3$) si deve al fatto che, per via della logica binaria, è il 2 e non il 10 il numero "re" dell'informatica.
Kilo	**$1000 = 10^3$**	**$1024 = 2^{10}$**		
Mega	**$1.000.000 = 10^6$**	**$1.048.576 = 2^{20}$**		
Giga	**$1.000.000.000 = 10^9$**	**$1.073.741.824 = 2^{30}$**		

Attualmente, le capacità "tipiche" delle varie memorie sono:

RAM	HARD DISK (HD)	CHIAVETTA USB	CD	DVD
Consigliati almeno 4 GB	Da 500 GB a 1 TB	Da 8 a 32 GB e più	650–700–800–870 MB	Da 4,7 GB in su

2b - QUANTE DIVERSE INFORMAZIONI SI POSSONO CODIFICARE CON UN BYTE?

In altre parole, avendo a disposizione 8 caselle con la possibilità di riempire ciascuna casella o con “0” o con “1”, quante sequenze fra loro distinte è possibile scrivere?

Facile: ogni Byte può essere interpretato come un numero binario a 8 cifre;
e numeri siffatti possono andare da 0 (00000000) **a 255** (11111111).
La risposta è dunque 255+1=256: con un Byte possiamo rappresentare fino a 256 diverse informazioni.
Osserviamo che è $256 = 2^8$.

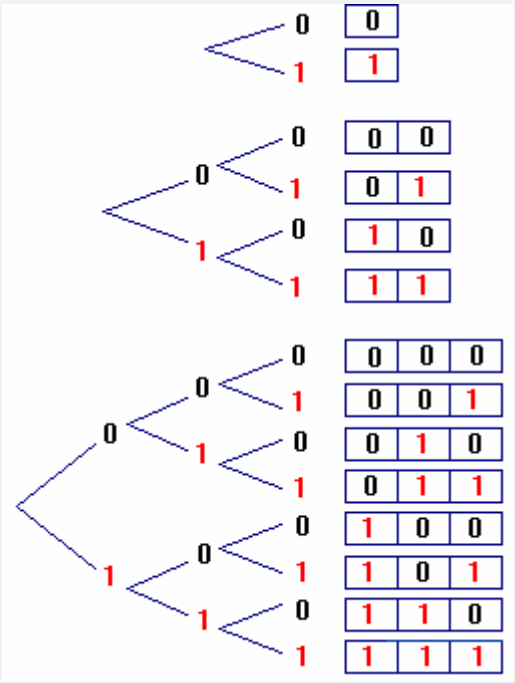
*Si sarebbe potuto anche ragionare in maniera **completamente diversa**.*

Ecco qui davanti a me la sequenza delle 8 caselle da riempire:

Per la prima casella, ho evidentemente due possibilità: 0 o 1.

Comunque io abbia scelto di riempire la 1ª casella,
per la 2ª mi si apre un ventaglio di 2 possibilità: 0 o 1.

Le possibilità, per quanto riguarda le **prime 2 caselle**,
sono espresse dal diagramma ad albero qui a destra:
$4 = 2^2$ possibilità (00, 01, 10, 11)



Comunque io abbia scelto il contenuto delle prime 2 caselle,
per riempire la 3ª mi si apre ancora un ventaglio di 2 possibilità.

Le possibilità, per quanto riguarda le **prime 3 caselle**,
sono espresse dal diagramma ad albero qui a destra:
$8 = 2^3$ possibilità (000, 001, 010, 011, 100, 101, 110, 111)

Ogni volta che penso a una casella in più, il numero di modi in cui è possibile riempire la sequenza di caselle considerate raddoppia per via del ventaglio di 2 possibilità che si apre.

A questo punto, è immediato concludere: **Byte = 8 caselle = 2^8 = 256 possibilità** .

Domanda: *quante informazioni diverse potranno essere codificate utilizzando 2 Byte? E 4 Byte? E 8 Byte?*

2c - UN MODO RAPIDISSIMO ED EFFICACISSIMO PER INDICARE UN BYTE

Nel sistema di numerazione ESADECIMALE, ossia in base SEDICI, le cifre sono:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
zero	uno	due	tre	quattro	cinque	sei	sette	otto	nove	dieci	undici	dodici	tredici	quattordici	quindici

Ma nel sistema BINARIO i numeri da zero a quindici
sono proprio quelli che sono rappresentabili con una quaterna di cifre 0, 1:

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
zero	uno	due	tre	quattro	cinque	sei	sette	otto	nove	dieci	undici	dodici	tredici	quattordici	quindici

E’ quindi possibile “riassumere” una sequenza di quattro bit (detta *nibble*),
interpretandola come numero binario e passando all’equivalente esadecimale!

0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**

Ma se **un *nibble* può essere “compresso” in una cifra esadecimale**,
allora **un *Byte* si potrà comprimere in una COPPIA di cifre esadecimali**!!!
Per cui, ad esempio,

- il Byte 0110 1110 potrà essere scritto, in breve, come 6E;
- il Byte 0011 0011 potrà essere scritto, in breve, come 33;
- la scrittura A8 potrà indicare il Byte 1010 1000;
- la scrittura 74 potrà indicare il Byte 0111 0100;
- ecc. ecc.

Risposta alla domanda del paragrafo 2b:
$2^{16} = 65\,536$,
$2^{32} = 4\,294\,967\,296$,
$2^{64} = 18\,446\,744\,073\,709\,551\,616$

2d - LE OPERAZIONI LOGICHE DEL MICROPROCESSORE

In Logica, il significato dei connettivi **ET, VEL, NON,** che ora indicheremo, all'inglese, con **AND, OR, NOT,** e a cui affiancheremo il connettivo di **"OR ESCLUSIVO"** o **XOR** (in pratica, l' **AUT** latino), è descritto dalle seguenti "tavole di verità":

p	q	p AND q
V	V	**V**
V	F	**F**
F	V	**F**
F	F	**F**

p	q	p OR q
V	V	**V**
V	F	**V**
F	V	**V**
F	F	**F**

p	NOT p
V	**F**
F	**V**

p	q	p XOR q
V	V	**F**
V	F	**V**
F	V	**V**
F	F	**F**

Se rappresentiamo "Vero" con 1 e "Falso" con 0, le tavole diventano:

p	q	p AND q
1	1	**1**
1	0	**0**
0	1	**0**
0	0	**0**

p	q	p OR q
1	1	**1**
1	0	**1**
0	1	**1**
0	0	**0**

p	NOT p
1	**0**
0	**1**

p	q	p XOR q
1	1	**0**
1	0	**1**
0	1	**1**
0	0	**0**

e definiscono altrettante vere e proprie "operazioni sui bit".

Ma mentre delle operazioni **and, or, not, xor** effettuate **sulle proposizioni** ci é ben chiaro il significato, che senso avranno e a quali finalità potranno servire le corrispondenti operazioni **sui bit**?

La sorprendente risposta è che **TUTTO** il funzionamento della CPU è sostanzialmente una manipolazione dei bit, effettuata tramite queste quattro operazioni logiche (e poche altre affini) !!!

Supponiamo ad esempio che la CPU debba sommare due numeri interi.
Preliminarmente, i due interi in gioco saranno stati codificati nel sistema binario, e immagazzinati in RAM. Ora la CPU li preleva dalla RAM e li colloca, rispettivamente, in due memoriette interne alla CPU stessa (registri). I due numeri sono in attesa di essere sommati.

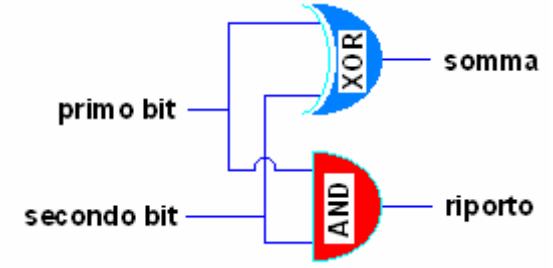
Ma quali sono le regole per la somma, quando gli addendi sono codificati in binario? Eccole:

a	b	somma	riporto
1	1	**0**	**1**
1	0	**1**	**0**
0	1	**1**	**0**
0	0	**0**	**0**

Ti segnalo, dal sito http://richardbowles.tripod.com
a cura di Richard Bowles,
un'interessante
Beginner's Guide to Digital Electronics
Clicca sulla freccia! ⇨

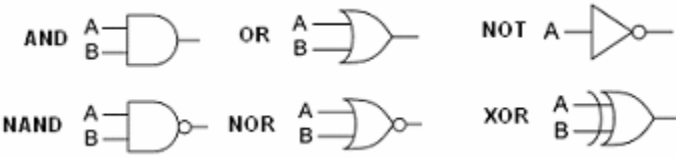
Scopriamo allora che la somma è data da **a XOR b**, mentre il **riporto** (*carry*) è dato da **a AND b** !!!

E siccome le tecnologie elettroniche consentono di realizzare con facilità microcomponenti in grado di ricevere in entrata (input) una coppia di bit, e di fornire in uscita (output) un terzo bit, in maniera da simulare le varie operazioni logiche, ecco che tramite una "porta logica" XOR e una "porta logica" AND è possibile realizzare il semplice dispositivo elettronico detto **semisommatore** (bit 1 = impulso elettrico presente, bit 0 = impulso elettrico assente):



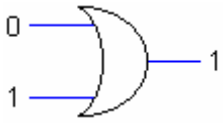
← Qui **a sinistra: un semisommatore** (half adder)

Sotto: simboli grafici per le varie "porte logiche"



Senza entrare nei particolari, si può comprendere che, disponendo e combinando in modo adeguato porte logiche appropriate, sarà possibile andare oltre, e realizzare un sommatore completo, in grado di ricevere in input due sequenze di bit, corrispondenti ad una coppia di numeri binari, e di fornire come output quella sequenza di bit, che corrisponde alla loro somma.

Ancora: supponiamo che un programma preveda che sia eseguito un dato comando, nel caso sia verificata ALMENO UNA fra due certe condizioni.
Ovviamente al momento opportuno la CPU attiverà una porta logica **OR** la quale riceverà in input una coppia di bit che rappresenteranno le due condizioni in gioco (11 se entrambe le condizioni sono verificate, altrimenti, a seconda dei casi, 01, 10 oppure 00). Se il bit in output sarà 1, il comando verrà eseguito, altrimenti no.



Sopra: una porta logica OR (Richard Bowles)