APPUNTI DI INFORMATICA

Per il biennio di liceo scientifico scienze applicate

Sommario

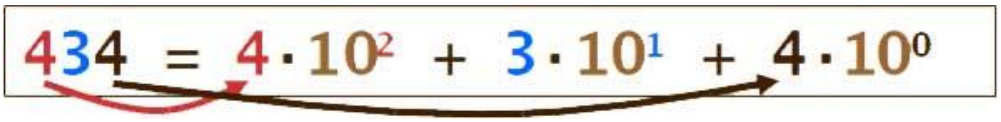
Vincenzo Galella, appunti di informatica.

Il calcolo binario

Numerazione Decimale

Il sistema si numerazione decimale, o sistema in base 10, è un sistema **posizionale** e utilizza le seguenti dieci cifre decimali: **0 1 2 3 4 5 6 7 8 9**

È un sistema posizionale perché una stessa cifra assume valori diversi (cioè ha un peso diverso) a seconda della posizione che occupa all'interno del numero



Il peso di una cifra e uguale alla base del sistema di numerazione, 10 in questo caso, elevata alla potenza uguale alla posizione della cifra nel numero; posizione che si incrementa da destra a sinistra a partire da 0

Numerazione Binaria

I computer usano invece numeri binari, cioè utilizzano in base 2; la base binaria usa solo due cifre diverse, 0 e 1.

Quindi, per scrivere un numero binario posso utilizzare solo i caratteri (o cifre) 0 e 1, utilizzare le altre cifre non avrebbe senso. Per tanto, 10101010011 è un numero binario, 2038 no.

Conversione Binario -> Decimale

Per convertire un numero binario in decimale, basta sommare il prodotto di ciascuna cifra con la potenza del 2 corrispondente alla posizione.

La posizione si calcola partendo da 0 per la cifra più a destra, e aumenta di 1 scorrendo verso sinistra. In altri termini, ciascuna cifra è in relazione con una precisa potenza del 2.

Ad esempio, convertiamo il numero binario 10110101

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	1	0	1	0	1

$$2^0 * 1 + 2^1 * 0 + 2^2 * 1 + 2^3 * 0 + 2^4 * 1 + 2^5 * 1 + 2^6 * 0 + 2^7 * 1 =$$

$$181$$

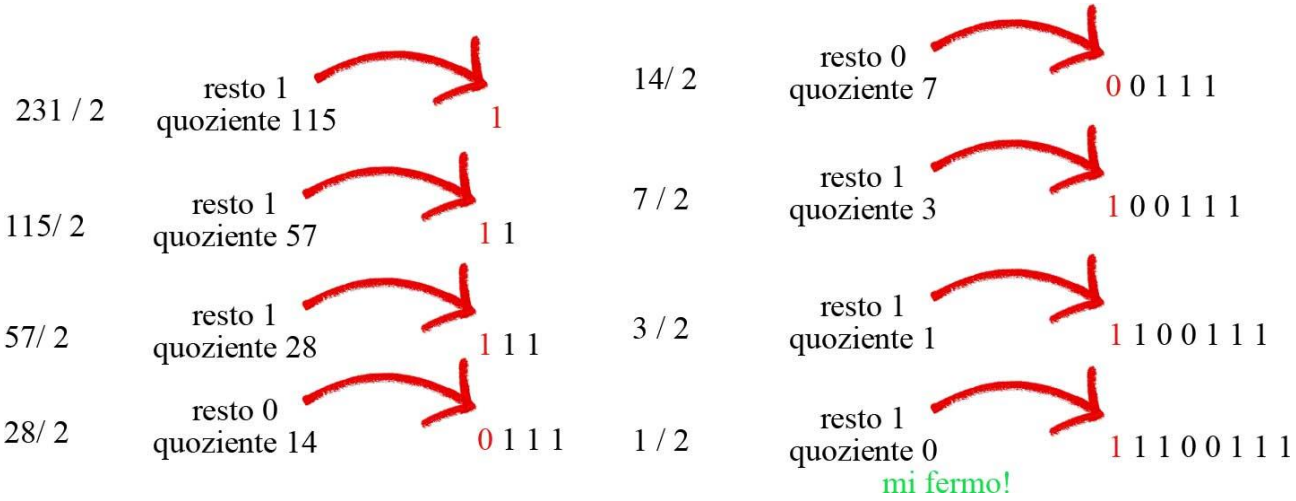
Vincenzo Galella, appunti di informatica.

Conversione Decimale -> Binario

Per convertire un numero decimale nella sua forma binaria, si procede come segue:

- si divide il numero decimale per due, finchè il quoziente diventa 0
- ogni divisione (fino al termine del procedimento) darà luogo ad un resto, che rappresenta la cifra binaria nella posizione corrispondente al numero di divisioni effettuate fino a quel momento

Ad esempio, convertiamo in binario il numero decimale 231

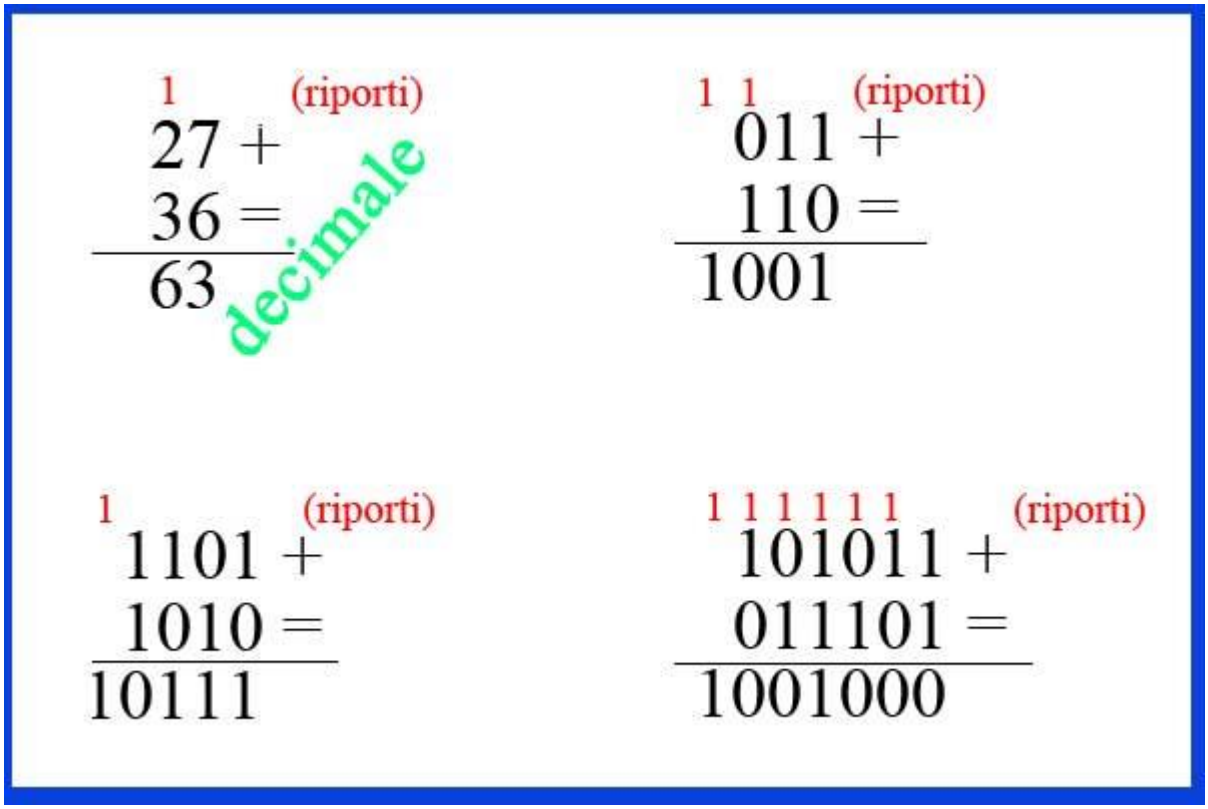


Vincenzo Galella, appunti di informatica.

Operazioni binarie

Somma tra numeri binari

La somma binaria segue le stesse regole della somma tra numeri decimali, di sotto sono riportati alcuni esempi pratici.

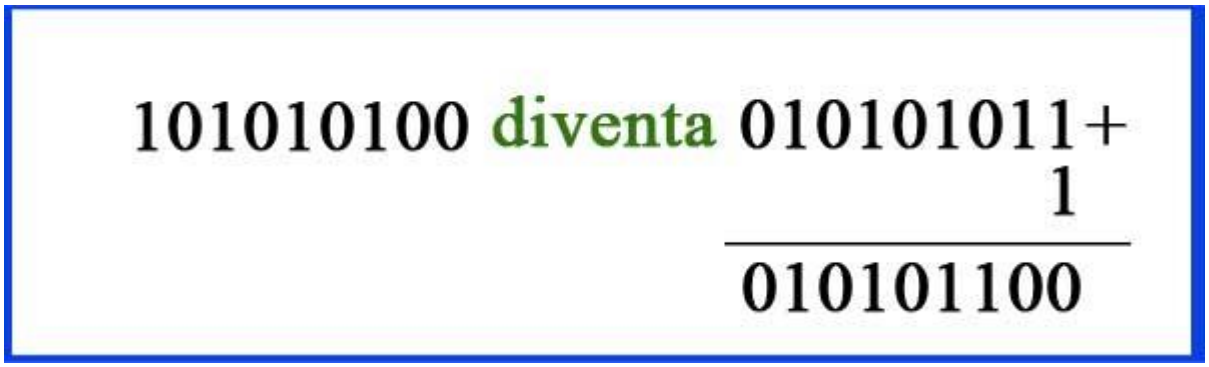


Il complemento a 1

Questa operazione, tipica del sistema binario, consiste semplicemente nell' "invertire" il valore delle cifre. Ovvero, gli "0" diventeranno "1" e viceversa.

Il complemento a 2

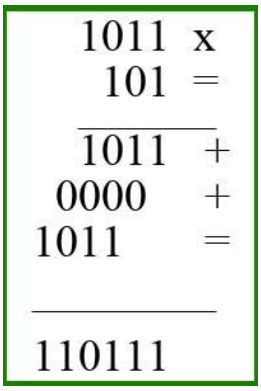
Questa operazione consiste nell' invertire le cifre binarie (come nel complemento a 1), sommando 1 al risultato.



La moltiplicazione binaria

Anche la moltiplicazione tra numeri binari segue le stesse regole della moltiplicazione tra numeri decimali, come riportato nell'esempio seguente.

Moltiplichiamo ad esempio 1011 (undici, in decimale) e 101 (cinque, in decimale), ottenendo 110111 (cinquantacinque, ovviamente!)



Ora moltiplichiamo 1011100 (92 in decimale) e 10 (che in decimale è 2).

E, poi, 1000101 (62 in decimale) e 10 (sempre 2, in decimale) e osserviamo i risultati

1011100 x 10 = 0000000 + 1011100 = 10111000	1000101 x 10 = 0000000 + 1000101 = 10001010

Entrambi i risultati non sono altro che il numero iniziale "con uno 0 in fondo", ovvero ciascuna cifra decimale ha incrementato la propria posizione. Nelle moltiplicazioni decimali cosa succede se moltiplico 423 x 10 ? Ottengo **4230**.

Questo perché moltiplico un numero in base dieci per una potenza della base ($10 = 10^1$), ovvero, matematicamente parlando

423 * 10 = $(4 * 10^2 + 2 * 10^1 + 3 * 10^0) * 10^1 = 4 * 10^{2+1} + 2 * 10^{1+1} + 3 * 10^{0+1} = 4230$

La stessa cosa accade se moltiplico 1011100 * 10, ovvero:

$$(1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0) * 2^1$$

Ora, cosa succede se moltiplico un numero binario per 100 (quattro decimale), ovvero 2^2 ?

1011100 x 100 = 0000000 + 0000000 1011100 = 101110000

Rappresentare i numeri interi binari negativi

Finora abbiamo visto come, con un byte, possiamo rappresentare 256 numeri interi (da 0 a 255). Poniamoci ora il problema di come si potrebbero rappresentare, sempre con un byte, numeri interi sia negativi sia positivi (l'idea è quella di rappresentarne metà positivi e metà negativi).

Esistono diverse possibilità a riguardo, ne esamineremo alcune tra le più significative.

Rappresentazione con "modulo" e "segno"

La prima tecnica consiste nel separare (concettualmente) il byte: il primo bit a sinistra rappresenterà il segno del numero binario, i restanti rappresenteranno il modulo, ovvero il valore assoluto.

Ad esempio, assumiamo che se il bit più a sinistra (quello dedicato al segno) vale 0, il numero è positivo, se è 1 allora è negativo.

Il numero 01010001 vale quindi +81, mentre il numero 10001110 vale -14.

Il principale difetto di questo sistema di rappresentazione è che lo 0 viene rappresentato due volte (00000000 e 10000000, ovvero +0 e -0).

I numeri rappresentati vanno da 11111111 (-127) a 01111111(+127), e sono per tanto 255 (e non 256 !!!).

Tecnica dell' "offset"

Una tecnica alternativa consiste nell'avere un numero decimale predefinito da sommare al numero binario: il risultato è il numero codificato.

Ad esempio, nel contesto di numeri binari di dimensione 1byte, stabilisco che l'offset valga -127.

A questo punto, per calcolare la forma decimale con segno di un numero binario, devo decodificarne il valore assoluto e sommare l'offset (ovviamente si tratterà di una sottrazione, perché sommo una quantità negativa).

Ad esempio, il numero binario 11001100 vale 204 **-127** = +77.

Invece, il numero 01110000 vale 112 **-127** = -15

Questo metodo consente di rappresentare i numeri che vanno da 00000000 (ovvero -127) a 11111111 (+128), lo 0 è rappresentato solo una volta (01111111) e osservando il bit più significativo (quello più a sinistra) posso subito dedurne il segno.

Ma, il principale problema, è che ogni volta che voglio decodificare un valore, devo fare un'ulteriore operazione (la somma dell'offset), e questo fa si che il metodo in questione sia poco pratico e mai utilizzato nei sistemi reali

Complemento a 2

Questo è il sistema più comunemente utilizzato per rappresentare i numeri binari con segno. I numeri positivi sono rappresentati esattamente come nel caso dei numeri senza segno, quelli negativi attraverso il loro complemento a 2.

Ad esempio la codifica del numero decimale 17 sarà 00010001, quella del numero -17 sarà: 11101111

00010001 (17 in binario) -> 11101110 (il suo complemento a 1) -> 11101111 (il suo complemento a 2)

In fase di decodifica, osservo il valore del bit più a sinistra:

se è 0, il numero è positivo, e mi basta decodificane il valore;

se è 1, il numero è negativo: riapplico il complemento a 2 per ottenerne il valore assouto.

Ad esempio:

01101101 : il bit a sinistra è 0, il numero è positivo e vale +109

11001111 : il bit a sinistra è 1, il numero è negativo. Riapplico il complemento a 2 e ottengo 00110001, ovvero 49, il numero vale quindi -49

Questa tecnica, inoltre, prevede la stessa codifica per i numeri decimali +0 e -0, infatti:

+0 = 00000000 OVVIO

-0 = complemento a 2 di (00000000) cioè 11111111 + 1 = 100000000 (attenzione, prendo solo gli 8 bit più a destra), quindi 00000000

Il numero più piccolo rappresentabile è 10000000, il cui complemento a 2 è 10000000, quindi -128.

Il numero più grande è 01111111, cioè +127.

Con questa tecnica posso quindi rappresentare 256 numeri diversi, da -128 a +127

Numerazione esadecimale

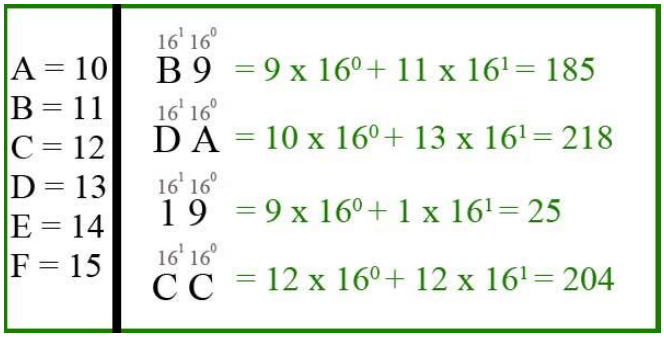
Questa rappresentazione utilizza una base 16, ovvero 16 cifre differenti e un calcolo che viene fatto sfruttando le proprietà delle potenze del 16.

Questa tecnica viene utilizzata perché rappresenta una forma più compatta della base 2 (infatti $16 = 2^4$), cioè con un carattere "esadecimale" rappresento 4 bit.

In particolare, le cifre sono : 0,1,2,3,4,5,6,7,8,9, A(che vale 10), B(che vale 11),C(12),D(13),E(14) e F(15).

Per decodificare in decimale un numero esadecimale basta moltiplicare il valore delle cifre esadecimali per 16 elevato alla posizione della cifra stessa.

In figura, alcuni esempi chiarificatori:



Somma esadecimale

La somma di due o più numeri esadecimali segue le stesse regole della somma binaria e della somma decimale. Se la somma di due o più cifre è maggiore della base (in questo caso 16) viene calcolato il quoziente e il resto del rapporto tra somma e base. Il resto rappresenta la cifra risultante, il quoziente rappresenta il "riporto".

Per tanto, la somma tra E5 e BD fa 1A2, un numero esadecimale a 3 cifre.

Vincenzo Galella, appunti di informatica.

Algebra booleana

L'algebra booleana è un particolare tipo di algebra definita su un insieme di due soli elementi, che possono essere rappresentati in modi diversi ma dal significato equivalente. I due valori in questione sono 0 e 1.

Esempi di possibili rappresentazioni dell'insieme:

Restano da definire le operazioni su questo insieme di valori: le tre principali operazioni sono

AND(congiunzione logica o prodotto logico)

OR(disgiunzione logica o somma logica)

NOT(negazione logica)

Definiamo ora il significato di queste operazioni.

Variabili booleane

Le variabili booleane sono elementi che possono assumere uno dei due possibili valori. Nella trattazione dell'argomento, i valori possibili saranno T e F, rispettivamente True(Vero) e False(Falso).

Facendo un paragone con l'algebra binaria, possiamo assumere che il valore T corrisponda al valore 1 binario, mentre il valore F corrisponde allo 0.

Operazioni booleane

NOT

Il not ($\neg$) è un operatore "unario", ovvero è un'operazione che ha un solo operando. In particolare, l'applicazione di questo operatore ad un valore booleano, restituisce "l'altro" valore.

Ad esempio not(T) = F e not(F) = T

L'operazione è simmetrica al complemento a 1 visto nell'algebra binaria, infatti, se T=1 e F=0,

infatti compl1(1) = 0, compl1(0) = 1

AND

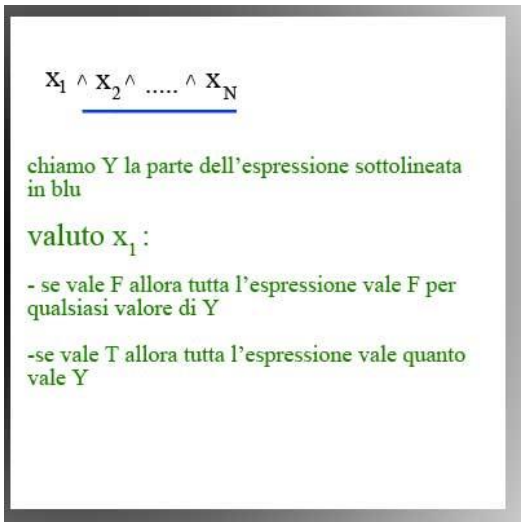
L'operatore and ($\wedge$) è un operatore binario, ovvero si applica a due operandi. Il significato è il seguente: l'and di due variabili logiche (o booleane) è T(vero) solamente se entrambe sono T(vere), altrimenti è F(falso).

Quindi :

$T \wedge T = T$ $T \wedge F = F$ $F \wedge T = F$ $F \wedge F = F$

Osserviamo cosa succede se ho n variabili booleane tutte in And tra di loro:

Non valuto subito tutte le variabili, perché non appena ne incontro una falsa posso assumere che tutta l'espressione sia falsa. Questa operazione è simmetrica alla moltiplicazione, infatti se io moltiplico N variabili binarie, se una di esse è 0 (cioè F in booleano) tutta l'espressione varrà 0.



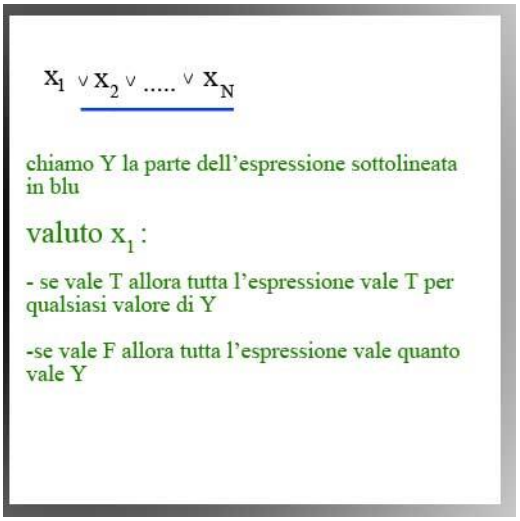
OR

L'operatore or ($\vee$) è un operatore binario, ovvero si applica a due operandi. Il significato è il seguente: l'or di due variabili logiche (o booleane) è T(vero) se almeno una è vera.

Quindi :

$T \vee T = T$ $T \vee F = T$ $F \vee T = T$ $F \vee F = F$

Osserviamo cosa succede se ho n variabili booleane tutte in Or tra di loro:



Non valuto subito tutte le variabili, perché non appena ne incontro una vera posso assumere che tutta l'espressione sia vera . Questa operazione è paragonabile ad una particolare addizione: immaginiamo che il risultato di una somma debba essere rappresentata con una sola cifra binaria (0 o 1). Posso assumere allora che, se la somma fa 0, la rappresento con 0, se invece fa qualsiasi valore maggiore o uguale a 1 la rappresento con 1.

In questo caso, l'analogia è evidente: sommo valori booleani, attribuendo 0 a F e 1 a T. Non appena sommo un 1, so già che il risultato della somma sarà 1.