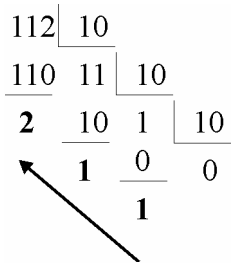
Conversione decimale → binario (continuazione)

La conversione di un numero decimale (es. 112) in binario si effettua tramite l'algoritmo della divisione, dividendo successivamente per 2 (base del sistema binario) il numero decimale da convertire e considerando i resti al contrario. Per scoprire il motivo per cui l'algoritmo funziona, analizziamo cosa succede nel caso on cui, invece di dividere per 2, si divide per 10 (base del sistema numerico decimale):



La prima divisione per 10 consente di "isolare" (stabilire quante sono) le unità, la seconda le decine, la terza le migliaia. Infatti, dopo la prima divisione (112/10), per definizione di divisione, si ha:

$$112 = 11 \cdot 10 + \mathbf{2}$$

In tal modo, le unità sono isolate poiché esse sono espresse tramite una sola cifra del sistema numerico (decimale). Un'ulteriore divisione per 10 darebbe, infatti, quoziente 0. Al contrario, le decine non lo sono poiché esse sono espresse dal numero 11 (due cifre). La seconda divisione, tuttavia, consente di isolare le anche decine:

$$112 = (1 \cdot 10 + 1) \cdot 10 + 2 = 1 \cdot 10^2 + \mathbf{1 \cdot 10^1} + 2 \cdot 10^0$$

Un'ulteriore divisione del quoziente (1) ottenuto nella seconda divisione per 10, consente di isolare anche le centinaia. In questo caso, essendo le centinaia espresse tramite una sola cifra del sistema, la terza divisione dà come risultato 0 e l'algoritmo termina. Se si continuasse a dividere ancora, il risultato sarebbe sempre 0 (cioè 0 migliaia, 0 decine di migliaia, ecc.).
La faccenda si può vedere anche in questi termini: se si divide successivamente un numero per 10, si ottengono progressivamente prima le unità, poi le decine, e così via:

1. 112/10 = 11.**2 (due unità)**
2. 11/10 = 1.**1** **(una decina)**
3. 1/10 =0.**1** **(una centinaia)**

Esattamente la stessa cosa succede se si divide per 2 invece che per 10. In questo caso, però si isolano le potenze di 2 e non le unità o le decine. E questo è il motivo per cui la conversione da decimale a binario si effettua con l'algoritmo della divisione considerando 2 come divisore.

Altri sistemi di numerazione

Ottale

Il sistema è basato sulle cifre: 0, 1, 2, 3, 4, 5, 6, 7.
Un numero ottale può essere rappresentato come somma di potenze di 8. La conversione da decimale a ottale si effettua sempre con l'algoritmo delle divisioni successive, dividendo per 8.

Esercizio
Convertire i seguenti numeri decimali in ottale:

1) 4
2) 10
3) 15

Esadecimale

Il sistema è basato sulle cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
Se si volesse trovare un corrispettivo numerico, questo potrebbe essere il seguente;

A=10
B=11
C=12
D=13
E=14
F=15

Un numero esadecimale può essere rappresentato come somma di potenze di 16. La conversione da decimale a esadecimale si effettua sempre con l'algoritmo delle divisioni successive, dividendo per 16.

Esercizio
Convertire i seguenti numeri decimali in esadecimale:

1) 10
2) 18
3) 30

Rappresentazione binaria dei numeri negativi

Fino a ora abbiamo visto come è possibile rappresentare i numeri naturali attraverso la rappresentazione binaria. Poiché i computer possono operare solo su due differenti simboli, è necessario adottare una "convenzione" per la rappresentazione dei numeri interi negativi. La convezione standard consiste nel considerare il bit più a sinistra per la rappresentazione del segno:

0 = '+'
1 = '-'

Esempio

1101 = -101 = -5

In tal modo, se si utilizzano *n* bit, è possibile rappresentare 2^n-1 numeri. Ad esempio, se *n*=2 è possibile rappresentare 2^4-1=3 numeri, essendo ridondante la rappresentazione del numero 0:

11 = -1; **1**0 = 0 ; **0**0 = 0; **0**1 = 1

Se *n*=8 è possibile rappresentare 2^8-1= 255 numeri:

11111111 = -127
11111110 = -126
…
10000000 = 0
00000000 = 0
…
01111110 = 126
01111111= 127

In generale con n bit si rappresentano i valori da -2^{n-1}-1 a 2^{n-1}-1.

Rappresentazione dei numeri positivi e negativi complemento a 2

Il complemento a due è il metodo più diffuso per la rappresentazione dei numeri interi in informatica. E' stata inventata negli anni '60 per evitare la presenza di due zeri (uno zero positivo e uno negativo) e per utilizzare tutte le possibili 2^n combinazioni che si possono ottenere con n bit.

Il bit iniziale (più a sinistra) del numero indica il segno mentre il modulo si ottiene invertendo il valore dei singoli bit e aggiungendo 1 al numero binario risultante.
Se n è il numero di bit utilizzati, si possono rappresentare 2^n numeri. Così, con 8 bit si possono rappresentare i numeri compresi tra -128 e +127.

Nota. Per complemento a 2 di un numero binario, si intende quel numero che, sommato al numero di partenza dia 0 come risultato. Infatti, in matematica il numero che manca a un altro numero per arrivare a ottenere un terzo numero viene detto complemento: il complemento di 4 per ottenere 7 é 3 (essendo 4+3=7) o, in altre parole, 4 complemento a 7 è uguale a 3. Il numero 2 in binario equivale allo 0, pertanto x_2 complemento a 2 è uguale a $-x_2$, essendo $x_2 + (-x_2) = 0$.

Per trovare il complemento a due di un numero binario se ne invertono, o negano, i singoli bit: si applica cioè l'operazione logica NOT. Si aggiunge infine 1 al valore del numero trovato con questa operazione.

Esempio
Rappresentazione del numero -5 con 8 bit in complemento a 2.
Si scrive innanzitutto la rappresentazione binaria del numero 5:

$$5_{10} = (0000\ 0101)_2$$

Si invertono i bit in modo che 0 diventi 1, e 1 diventi 0:

1111 1010

A questo punto abbiamo ottenuto il "complemento a uno" del numero 5. Per ottenere il complemento a due aggiungiamo 1:

1111 1011

Esempio: Sommare i numeri 5 = 0000 0101 e -5 = 1111 1011.

```
 0000 0101 +
 1111 1011 =
 ----------
1 0000 0000  .
```

Si ignora l'overflow poiché abbiamo scelto una rappresentazione a 8 bit e quindi il risultato è 0=0000 0000.

Esempio: Sommare i numeri 5 = 0000 0101 e 1 = 0000 0001.

0000 0101 +
0000 0001 =

0000 0110 .

Il risultato è 6=0000 0110.

Esempio: Sommare i numeri 5 = 0000 0101 e -1.
Innanzitutto si scrive il complemento a 2 del numero binario che rappresenta il numero -1:

1_{10}=0000 0001; si invertono i bit: 1111 1110; si somma 1: 1111 1111 = $-1_{\text{complemento a 2}}$

Si fa la somma:

0000 0101 +
1111 1111 =
---
1 0000 0100 .

Si ignora l'overflow e il risultato è 4=0000 0100

Osservazione
Il complemento a due di un numero negativo ne restituisce il modulo (valore assoluto). Invertendo i bit della rappresentazione del numero -5 (sopra) otteniamo: 0000 0100; Aggiungendo 1 otteniamo: 0000 0101, che è appunto la rappresentazione del numero +5 in forma binaria.
Proviamo col numero -1:
1111 1111 = -1 (complemento a 2); si invertono i bit: 0000 0000; si somma 1: 0000 0001=1.

Osservazione
Si noti che il complemento a due dello zero è zero stesso: invertendone la rappresentazione si ottiene un byte di 8 bit pari a 1, e aggiungendo 1 si ritorna a tutti 0 (l'overflow viene ignorato).

0000 0000 [inversione]→	1111 1111 [somma 1]→	1111 1111 + 0000 0001 = _____ 1 0000 0000	[si ignora l'overflow]→ 0000 0000

Inoltre, senza entrare nei dettagli, il complemento a 2 consente di operare le operazioni di addizione e sottrazione in modo più efficiente rispetto alle altre rappresentazioni, motivo per cui è quella comunemente utilizzata nei calcolatori elettronici per i numeri interi.

Approfondimento sul Internet
Chi ne volesse sapere di più può dare un'occhiata al seguente link:
http://it.wikipedia.org/wiki/Complemento_a_due.

Rappresentazione di numeri frazionari

I numeri interi sono molto utili se vogliono rappresentare oggetti numerabili. Se però vogliamo misurare qualcosa abbiamo bisogno di numeri con la virgola decimale, cioè con espressioni in cui compaiono frazioni decimali.

E' utile introdurre il concetto di precisione. Quando diciamo che una misura é precisa possiamo farlo in due modi: in modo assoluto e in modo relativo. In modo assoluto diremo, ad esempio, che abbiamo misurato le pareti di una stanza con la precisione di un centimetro, in questo caso l'errore massimo é assoluto e vale un centimetro. Una misura viene detta precisa in modo relativo quando ci interessa fare un errore inferiore ad una certa parte della grandezza da misurare: ad esempio misuriamo la velocità di un automobile con la precisione del 10% se l'automobile va a 150 Km all'ora l'errore é di 15 Km/h. Se aggiungiamo l'errore di 15 Km/h in più otteniamo 165 Km/h. Se togliamo l'errore di 15 Km/h otteniamo 135 Km/h. La cifra che comunque non varia nel primo (sovrastima della velocità) e nel secondo caso (sottostima della velocità) é la prima cifra che viene quindi detta cifra esatta.

Rappresentazione di numeri frazionari in "virgola mobile" (Floating point)

Nei numeri a virgola mobile la informazione relativa alle cifre significative viene archiviata con un gruppo di bit che viene detto mantissa. Il numero di zeri da aggiungere é conservato separatamente e viene detto esponente. Sia la mantissa che l'esponente sono quindi dei numeri interi e vengono memorizzati con le regole viste in precedenza per i numeri interi. Se x è il numero che si vuole rappresentare, si utilizza la seguente convenzione:

$$x = \pm\ 0.m \times \mathrm{B}^{e}$$

dove m è detta mantissa, e è detto esponente, mentre B rappresenta la base del sistema numerico. Ad esempio, il numero 123.45 si rappresenta nel sistema numerico decimale nel seguente modo:

$$123.45 = 0.12345 \times 10^{3}$$

In questo caso la mantissa, m, è 123, l'esponente, e, è 3 e la base, B, è 10 (sistema decimale).
Nel caso in cui si scelga il sistema binario e si voglia scrivere il numero 101010000, si ha:

$$101010000 = 0.10101 \times 10^{01001}$$

dove $m=10101_2$, $B=10_2=2_{10}$ ed $e=01001_{C2}=9_{10}$.

Il problema di questo tipo di rappresentazione è la perdita di precisione. Supponiamo di operare nel sistema decimale e di voler utilizzare 4 cifre per m e 1 cifra per e. La rappresentazione del numero 123.45 non consente di preservare la seconda cifra decimale, perdendo in precisione:

$$123.45 = 0.1234 \times 10^{3}$$

Negli standard attualmente usati (IEEE 754) i numeri a virgola mobile sono di due tipi:

- 32 bit - singola precisione: 24 bit per la mantissa e 8 bit per l'esponente
- 64 bit - doppia precisione: 53 bit per la mantissa e 11 bit per l'esponente

Se vogliamo trasformare la precisione espressa in bit in precisione espressa in numero di cifre decimali abbiamo:

- singola precisione: 6 cifre decimali
- doppia precisione : 15 cifre decimali.

Questo risultato si ottiene partendo dal fatto che 10 bit servono per 3 cifre decimali. Quindi 24 bit danno circa 6 cifre decimali.
La dimensione massima dei numeri rappresentabili é di $10^{\pm 38}$ e di $10^{\pm 308}$ che si ottiene dalla precisione in bit sull'esponente.

Rappresentazione di numeri frazionari in "virgola fissa"

Il numero frazionario è rappresentato come una coppia di numeri interi: la parte intera e la parte decimale. Se si utilizzano 8 bit sia per la parte intera che per la parte decimale, il numero 12.54 è codificato come segue:

$$12.52 \rightarrow (12, 52) \rightarrow (00001100, 00110100)$$

Questo tipo di codifica è stata abbandonata.

La Codifica dei Caratteri

I caratteri sono quei simboli alfabetici o numerici che possono essere introdotti da una tastiera. Un carattere viene introdotto nel computer a partire da un dispositivo elettronico, la tastiera. Vi possono essere tastiere diverse per ogni paese in quanto alcuni paesi usano tipi di caratteri diversi: ad esempio in Italia si usano anche i caratteri corrispondenti alle vocali accentate in modo grave o acuto come à,à,é,è,í,ì,ó,ò,ú,ù. In Francia o Spagna si usa ad esempio la cedilla cioé il carattere ç e nelle tastiere di quei paesi é presente un tasto relativo.
Se un calcolatore deve funzionare in questi paesi l'unica cosa che verrà sostituita é la tastiera mentre ovviamente il computer deve essere costruito per poter funzionare indipendentemente dal paese. Ma quanti bit sono necessari per rappresentare un carattere ?. Su questo aspetto negli anni 60 e 70 vi é stata molta discussione. Alcuni costruttori suggerivano 6 bit, altri 7 infine si sono affermati gli 8 ed i 16 bit. Un numero di bit uguale a 6 fornisce solo 64 combinazioni. Con 64 combinazioni si potevano memorizzare caratteri alfabetici e numerici, i vari segni di interpunzione come la virgola il punto ma non si potevano inserire i caratteri minuscoli o i caratteri speciali dei vari paesi. La soluzione a 7 bit con le sue 128 possibilità permette di inserire tutti i tipi di carattere maiuscoli e minuscoli ma non i caratteri dei vari paesi. La soluzione più diffusa attualmente é quella a 8 bit con la quale si possono rappresentare 256 caratteri. Con questa scelta è possibile inserire anche caratteri speciali che servono per funzioni particolari come le comunicazioni o per rappresentare segni grafici speciali. Un secondo problema é come disporre i caratteri tra le varie combinazioni ad esempio la combinazione 33 (00100001) corrisponde normalmente alla A (a maiuscola), ma chi lo stabilisce? Ovviamente serve uno standard cioè un modo uniforme di considerare i caratteri. In caso contrario se alla A un computer associa la combinazione 65 e alla B la combinazione 66 ed un altro computer fa l'opposto, quando trasferiamo un testo da un computer all'altro le A diventano B e viceversa, ad esempio BABA diventa ABAB.

Lo standard oggi più diffuso é quello **ASCII (American Standard Code for Information Interchange) a 7 bit**. Lo standard ASCII non dà purtroppo una definizione precisa dei caratteri dalla combinazione 128 fino alla 255 e vi sono varie possibilità una di queste la codifica **ASCII ESTESA a 8 bit**.

00000000	Null	00100000	Spc	01000000	@	01100000	`
00000001	Start of heading	00100001	!	01000001	A	01100001	a
00000010	Start of text	00100010	"	01000010	B	01100010	b
00000011	End of text	00100011	#	01000011	C	01100011	c
00000100	End of transmit	00100100	$	01000100	D	01100100	d
00000101	Enquiry	00100101	%	01000101	E	01100101	e
00000110	Acknowledge	00100110	&	01000110	F	01100110	f
00000111	Audible bell	00100111	'	01000111	G	01100111	g
00001000	Backspace	00101000	(	01001000	H	01101000	h
00001001	Horizontal tab	00101001	)	01001001	I	01101001	i
00001010	Line feed	00101010	*	01001010	J	01101010	j
00001011	Vertical tab	00101011	+	01001011	K	01101011	k
00001100	Form Feed	00101100	,	01001100	L	01101100	l
00001101	Carriage return	00101101	-	01001101	M	01101101	m
00001110	Shift out	00101110	.	01001110	N	01101110	n
00001111	Shift in	00101111	/	01001111	O	01101111	o
00010000	Data link escape	00110000	0	01010000	P	01110000	p
00010001	Device control 1	00110001	1	01010001	Q	01110001	q
00010010	Device control 2	00110010	2	01010010	R	01110010	r
00010011	Device control 3	00110011	3	01010011	S	01110011	s
00010100	Device control 4	00110100	4	01010100	T	01110100	t
00010101	Neg. acknowledge	00110101	5	01010101	U	01110101	u
00010110	Synchronous idle	00110110	6	01010110	V	01110110	v
00010111	End trans. block	00110111	7	01010111	W	01110111	w
00011000	Cancel	00111000	8	01011000	X	01111000	x
00011001	End of medium	00111001	9	01011001	Y	01111001	y
00011010	Substitution	00111010	:	01011010	Z	01111010	z
00011011	Escape	00111011	;	01011011	[	01111011	{
00011100	File separator	00111100	<	01011100	\	01111100	|
00011101	Group separator	00111101	=	01011101	]	01111101	}
00011110	Record Separator	00111110	>	01011110	^	01111110	~
00011111	Unit separator	00111111	?	01011111	_	01111111	Del

Per i caratteri nei moderni sistemi operativi è utilizzata la codifica **UNICODE a16 bit**. Il numero di possibili simboli rappresentabili é 65536, e si possono utilizzare anche per rappresentare caratteri ideografici come ad esempio il Kanij dei giapponesi.

Nota. Le cifre 0..9 rappresentate in ASCII o UNICODE sono simboli o caratteri e NON quantità numeriche. Non pertanto possibile usarle per indicare quantità e per le operazioni aritmetiche.

Codifica delle immagini

Le immagini si suddividono in **raster** e **vettoriali**.

Le immagini raster sono idealmente suddivise in punti (pxel) disposti su una griglia a righe orizzontali e verticali. In origine, in un'immagine raster di tipo **bitmap** (mappa di bit) era possibile rappresentare solo due colori: bianco (bit 0) e nero (bit 1). Tuttavia, l'accezione attualmente in uso prevede diverse profondità di colore per i pixel dell'immagine. Con 8 bit è possibile rappresentare 256 colori (es. le tonalità di grigio), con 16 più di 32000 colori, e così via.
Essendo le immagini un insieme di punti colorati disposti su una griglia esse sono rappresentate con un certo livello di approssimazione, o meglio, di risoluzione. La risoluzione è data dal prodotto delle colonne e delle righe della griglia. Ad esempio, quando si parla di un'immagine alla risoluzione di 640×480 pixel, significa che i pixel totali sono appunto 640·480=307200 e che essi sono disposti su una griglia di 640 colonne (larghezza) e 480 righe (altezza).
Un'immagine di 640×480 pixel con 256 colori occupa 640×480×8 = 2457600 bit = 307200 byte = 307.2 Kbyte.
E' possibile risparmiare spazio nella rappresentazione delle immagini. Ad esempio, le aree dello stesso colore si possono rappresentare in modo "abbreviato": si specifica il colore e la lista delle coordinate dei pixel di quel colore. Esistono vari formati di codifica bitmap compressa: gif, jpg, pict, tiff e altri. In genere, è possibile passare da un formato ad un altro.

Le immagini possono essere anche rappresentate in forma "vettoriale". In questo caso si specificano le istruzioni su come disegnare l'immagine, per esempio del tipo:

…
colore (255,128,0)
spessore 1
alpha = 1
linea dal punto (10,12) al punto (20,30)
colore (0,0,125)
linea dal punto (20,30) al punto (20,35)
…

Le immagini vettoriali sono comunque tradotte in formato bitmap al momento della visualizzazione (si ricordi che anche il monitor è una griglia di pixel). Il vantaggio di questi formati è che non si perde di qualità nello "scaling"(zoom) dell'immagine. Il formato è particolarmente idoneo per immagini di tipo geometrico (per esempio progetti ingegneristici) e generalmente richiede poco spazio.
Il formato più diffuso è il PostScript (ps, eps), anche per la stampa dei testi. Un altro esempio di formato vettoriale è il cdr di CorelDraw.

Codifica dei filmati

Sono sequenze di immagini compresse: (ad esempio si possono registrare solo le variazioni tra un fotogramma e l'altro).Esistono vari formati (compresi i suoni):
avi (microsoft)
mpeg (il piu' usato)
quicktime mov (apple)

Codifica dei suoni

L'onda sonora viene misurata (campionata) a intervalli regolari. Minore è l'intervallo di campionamento, maggiore è la qualità del suono. Per i CD musical si hai: 44000 campionamenti al secondo, 16 bit per campione.