Dispensa del corso di Informatica

Il computer e i dati

La Memoria e i digit

Si è visto che è possibile utilizzare sistemi diversi da quello decimale per rappresentare i numeri e si è anche visto che è possibile effettuare operazioni. Come si può facilmente immaginare oltre ai numeri positivi è possibile rappresentare i negativi e i numeri reali, in notazione con cifre in virgola. E persino le frazioni.

Tra le possibili basi tra cui scegliere, il sistema di elaborazione elettronica frutta il sistema binario, spesso abbreviato in digitale. In effetti l'uso delle cifre 0 e 1 consente non solo di rappresentare cifre numeriche, ma anche valori logici (booleani), indirizzi e altri segnali.

Il bit

Fonte: http://it.wikipedia.org/wiki/Bit

In informatica e in teoria dell'informazione, la parola bit ha tre significati molto diversi, a seconda del contesto in cui rispettivamente la si usa:

- un bit è l'unità di misura dell'informazione (dall'inglese "**binary information unit**"), definita come la quantità minima di informazione che serve a discernere tra due possibili eventi equiprobabili.
- un bit è una cifra binaria, (in inglese "**binary digit**") ovvero uno dei due simboli del sistema numerico binario, classicamente chiamati zero (0) e uno (1).
- il bit è l' espressione della **qualità del colore** di un apparecchio elettronico. In questo campo la misura raddoppia sempre(es. 8bit 16bit 32bit 64bit)

Il bit come quantità di informazione

Questo concetto di bit è stato introdotto dalla teoria dell'informazione di Claude Shannon nel 1948, ed è usato nel campo della compressione dati e delle trasmissioni numeriche.

Intuitivamente, equivale alla scelta tra due valori (sì/no, vero/falso, acceso/spento), quando questi hanno la stessa probabilità di essere scelti. In generale, per eventi non necessariamente equiprobabili, la quantità d'informazione di un evento rappresenta la "sorpresa" nel constatare il verificarsi di tale evento; per esempio, se un evento è certo, il suo verificarsi non sorprende nessuno, quindi il suo contenuto informativo è nullo; se invece un evento è raro, il suo verificarsi è sorprendente, quindi il suo contenuto informativo è alto.

Il bit come cifra binaria

In questo contesto il bit rappresenta l'unità di definizione di uno **stato logico**. Definito anche unità elementare dell'informazione trattata da un elaboratore. La rappresentazione logica del bit è rappresentata dai soli valori {0, 1}. Ai fini della programmazione è comune raggruppare sequenze di bit in entità più vaste che possono assumere valori in intervalli assai più ampi di quello consentito da un singolo bit. Questi raggruppamenti contengono generalmente un numero di stringhe binarie pari a una potenza binaria, pari cioè a $\mathbf{2^n}$; il più noto è il **byte** (chiamato anche ottetto), corrispondente a **8 bit**, che costituisce l'unità di misura più utilizzata in campo informatico. I raggruppamenti più diffusi sono:

nibble	4 bit	la metà di un byte	quantità di informazioni possibili	2^4	16
byte	8 bit	unità fondamentale	quantità di informazioni possibili	2^8	256
word	16 bit	parola elementare	quantità di informazioni possibili	2^{16}	65536
double word	32 bit	parola doppia	quantità di informazioni possibili	2^{32}	> 4 miliardi
quad word	64 bit	parola quadrupla	quantità di informazioni possibili	2^{64}	> 18 mld mld

Il bit quindi indica l'unità fondamentale di rappresentazione binaria, ed è facilmente rappresentabile con sistemi elettronici, magnetici o ottici per indicare la quantità 0 o 1.

Byte, word e sistemi a molti bit

Byte

Fonte: http://it.wikipedia.org/wiki/Byte

Un byte, il cui nome deriva dalla parola inglese **bite** (boccone, morso), coniato anche per assonanza col termine "bit" ma rinominato per evitare confusioni accidentali di pronuncia con questo, è una sequenza di bit, il cui numero dipende dall'implementazione fisica della macchina sottostante.

Storicamente, un byte era il numero di bit utilizzati per codificare un "singolo carattere di testo" in un computer, ed è perciò divenuto l'elemento base dell'indirizzabilità nelle architetture dei computer e come unità di misura delle capacità di memoria.

Dal 1964 il byte è tipicamente formato da 8 bit[4], ed è pertanto in grado di assumere 2^8 = 256 possibili valori (da 0 a 255). Gli informatici di lingua francese utilizzano il termine octet (ovvero ottetto), sebbene il termine venga utilizzato in inglese per denotare una generica sequenza di otto bit.

È possibile immaginare il byte come una sequenza di 8 bit impacchettati come in un filare di valori binari:

Byte =	1	0	1	1	0	0	1	0

Multipli del byte

Prefissi SI			Prefissi Binari		
Nome	Simbolo	Multiplo	Nome	Simbolo	Multiplo
Kilobyte	KB	10^3	Kibibyte	KiB	2^{10}
Megabyte	MB	10^6	Mebibyte	MiB	2^{20}
Gigabyte	GB	10^9	Gibibyte	GiB	2^{30}
Terabyte	TB	10^{12}	Tebibyte	TiB	2^{40}
Petabyte	PB	10^{15}	Pebibyte	PiB	2^{50}
Exabyte	EB	10^{18}	Exbibyte	EiB	2^{60}
Zettabyte	ZB	10^{21}	Zebibyte	ZiB	2^{70}
Yottabyte	YB	10^{24}	Yobibyte	YiB	2^{80}

Bit, indirizzi e dati

La memoria del computer non è infinita; pertanto è indispensabile dividerla in porzioni facilmente gestibili. All'inizio dell'era informatica la quantità sufficientemente ampia, ma non troppo grande, di porzione informativa venne stabilita in 8 bit. Con 8 bit è possibile rappresentare 256 combinazioni diverse ovvero:

0	=	0	0	0	0	0	0	0	0
1	=	0	0	0	0	0	0	0	1
2	=	0	0	0	0	0	0	1	0
3	=	0	0	0	0	0	0	1	1
4	=	0	0	0	0	0	1	0	0
	=								
254	=	1	1	1	1	1	1	1	0
255	=	1	1	1	1	1	1	1	1

È possibile passare da una rappresentazione di 8 zeri a rappresentazioni con zeri e uni fino alla rappresentazione con tutti uni. Tradizionalmente la rappresentazione di tutti zeri indica il numero zero, mentre quella con tutti uni indica il numero 255, il massimo possibile.

gli scienziati si posero quindi il problema di dover rappresentare dentro la memoria di un computer i simboli del'alfabeto inglese dalla lettera A alla lettera Z, maiuscole. Per risolvere questo problema si pensò di utilizzare una codifica, ovvero una corrispondenza tra una combinazione di bit (nel byte) e la lettera corrispondente. Per esempio si decise che la lettera maiuscola A dovesse corrispondere alla sequenza **`01000001`**. Le lettere maiuscole successive venivano enumerate conseguentemente:

A	=	0	1	0	0	0	0	0	1	= 65
B	=	0	1	0	0	0	0	1	0	= 66
C	=	0	1	0	0	0	0	1	1	= 67
D	=	0	1	0	0	0	1	0	0	= 68
E	=	0	1	0	0	0	1	0	1	= 69
...	...									... ...
Y	=	0	1	0	1	1	0	0	1	= 89
Z	=	0	1	0	1	1	0	1	0	= 90

la codifica sopra accennata prese il nome di codice ASCII (acronimo per American Standard Code for Information Interchange (ovvero Codice Standard Americano per lo Scambio di Informazioni), che si può pronunciare indifferentemente asci (versione italiana) o askey (anglosassone).

Nella figura seguente (wikipedia) è illustrato come ogni carattere sia codificato con una combinazione di 7 bit di cui i primi 3 bit (b_7, b_6, b_5) indicano la colonna e i successivi 4 bit (b_4, b_3, b_2, b_1) indicano la riga;

b_7 → b_6 → b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Rappresentazione numerica

Come abbiamo visto i valori 0 e 1 possono rappresentare qualsiasi informazione; gruppi di bit formano codifiche di informazioni articolate, per cui è possibile memorizzarle o inviarle con significati diversi.

Per esempio:

0 1 0 0 0 0 0 1	Interpretazione numerica	65
0 1 0 0 0 0 0 1	Codifica ASCII	"A"
0 1 0 0 0 0 0 1	Interpretazione booleana	F V F F F F F V

Ma in effetti esistono molte altre possibilità di interpretare e decodificare il medesimo valore per adeguarsi a regole diverse. Una delle varianti riguarda la rappresentazione del segno.

Se volessimo rappresentare il segno esiste una prima possibilità che è quella di utilizzare il primo bit come segno (0 vale + e 1 vale -) e quindi ottenere i seguenti valori:

0 0 0 0 0 0 0 1	+1	Lo zero iniziale è il segno positivo, le altre cifre il valore assoluto
1 0 0 0 0 0 0 1	-1	Lo zero iniziale è il segno negativo, le altre cifre il valore assoluto
0 1 0 0 0 0 0 1	+65	Vedi sopra
1 1 0 0 0 0 0 1	-65	Vedi sopra
0 0 0 0 0 0 0 0	+0	Vedi sopra, ma lo zero non ha segno!
1 0 0 0 0 0 0 0	-0	Vedi sopra, ma lo zero non ha segno!

Complemento a due

Il sistema sopra illustrato non è efficiente per almeno due motivi:

1. lo zero impiega due diverse codifiche, creando confusione e spreco di codifiche;
2. le operazioni algebriche di somma e sottrazione richiedono procedimenti più complicati del necessario

e per questo gli scienziati dell'informazione hanno elaborato sistemi alternativi.

Fonte: http://it.wikipedia.org/wiki/Complemento_a_due

Il complemento a due (in inglese *two's complement*) è il metodo più diffuso per la rappresentazione dei numeri negativi in informatica. L'espressione complemento a due viene spesso usata impropriamente per indicare l'operazione di negazione (cambiamento di segno) nei computer che usano questo metodo. La sua enorme diffusione è data dal fatto che i circuiti di addizione e sottrazione non devono esaminare il segno di un numero rappresentato con questo sistema per determinare quale delle due operazioni sia necessaria, permettendo tecnologie più semplici e maggiore precisione; si utilizza un solo circuito, il sommatore, sia per l'addizione che per la sottrazione.

Col complemento a due, il bit iniziale del numero (quello più a sinistra) ha peso negativo o positivo; da questo consegue che tutti i numeri che cominciano con un "1" sono numeri binari negativi, mentre tutti i numeri che cominciano con uno "0" sono numeri binari positivi.

Si può ottenere il valore assoluto di un numero binario negativo, prendendo il complementare (invertendo il valore dei singoli bit) e aggiungendo 1 al numero binario risultante.

Esempio

Consideriamo il numero binario di otto bit **10010110**

Poiché il primo bit è un 1 allora il numero è negativo

Per sapere quanto vale il numero dobbiamo anzitutto fare il complemento del numero dato ovvero 01101001 e quindi sommare +1 al risultato ottenendo: **01101010**

A questo punto valutiamo il numero nel modo consueto e otteniamo: **106**

Quindi il numero 10010110 in complemento a due vale **-106**

Con questo sistema un numero binario di n cifre può rappresentare i numeri interi compresi fra -2^{n-1} e $+2^{n-1}-1$.

Esempio

I sistemi binari più diffusi permettono di rappresentare i seguenti numeri interi:

8 bit	$-2^{7} = -128$	$+2^{7}-1 = +127$
16 bit	$-2^{15} = -32768$	$+2^{15}-1 = +32767$
32 bit	$-2^{31} = -2147483648$	$+2^{31}-1 = +2147483647$
64 bit	$-2^{63} = -9223372036854775808$	$+2^{63}-1 = +9223372036854775807$

E così via.

Questo metodo consente di avere un'unica rappresentazione dello zero (quando tutti i bit sono zero, eliminando così la ridondanza dello zero che si verifica con la rappresentazione in modulo e segno), e di operare efficientemente addizione e sottrazione sempre avendo il primo bit a indicare il segno.

Caratteristiche

Questo metodo di rappresentazione ha notevoli vantaggi, soprattutto per effettuare somme e differenze: in pratica ai numeri viene anteposto un bit di valore zero; se poi il numero è negativo è necessario convertirlo in complemento a 2: per farlo è sufficiente leggere il numero da destra verso sinistra e invertire tutte le cifre a partire dal primo bit uguale a 1 (escluso).

Per fare un esempio:

$$-12_{10} = -01100_2 = 10100_{CA2}$$

Come è possibile notare seguendo questo metodo il primo bit diventa automaticamente il bit del segno (come per il metodo precedente). Viene però risolto il problema dell'ambiguità dello 0 (in complemento a 2 00000 e 10000 hanno significati diversi) e vengono enormemente facilitate le operazioni di somma e differenza, che si riducono alla sola operazione di somma: per spiegare meglio basta fare un esempio:

$$5_{10}-10_{10} = 5_{10}+(-10)_{10} = 0101_2-1010_2 = 00101_{CA2}+10110_{CA2} = 11011_{CA2} = -00101_2 = -5_{10}$$

Per rappresentare l'opposto di un numero binario in complemento 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

Facciamo un esempio rappresentando il numero -5 con 8 bit in complemento a 2. Partiamo dalla rappresentazione in binario del numero 5:

$$\mathbf{+5_{10} = 0000\ 0101_2}$$

La prima cifra è 0, quindi il numero è sicuramente positivo. Invertiamo i bit: 0 diventa 1, e 1 diventa 0:

```
1111 1010
```

A questo punto abbiamo ottenuto il complemento a uno del numero 5; per ottenere il complemento a due aggiungiamo 1 a questo numero:

$$\mathbf{-5_{10} = 1111\ 1011_2}$$

Il risultato è un numero binario con segno che rappresenta il numero negativo -5 secondo il complemento a due. Il primo bit, pari a 1, evidenzia che il numero è negativo.

Lo stesso procedimento permette di ottenere un numero positivo partendo da un numero negativo.

Esempio

Per invertire il numero negativo -5 con 8 bit in complemento a 2, si parte dal numero negativo:

$$\mathbf{-5_{10} = 1111\ 1011_2}$$

La prima cifra è 1, quindi il numero è sicuramente negativo. Poi invertiamo i bit:

```
0000 0100
```

A questo punto aggiungiamo 1 a questo numero, ottenendo:

$$\mathbf{+5_{10} = 0000\ 0101_2}$$

Il risultato è un numero binario con segno che rappresenta il numero positivo **+5** in complemento a due. Il primo bit è zero ovvero il numero è positivo.

Addizione e sottrazione

Operare l'addizione di due interi rappresentati con questo metodo non richiede processi speciali se essi sono di segno opposto, e il segno viene determinato automaticamente. Facciamo un esempio addizionando 15 e -5:

Esempio

```
  0000 1111  (15)
+ 1111 1011  (-5)
     ==========
  0000 1010  (10)
```

Questo processo sfrutta la lunghezza fissa di 8 bit della rappresentazione: viene ignorato un riporto di 1 che causerebbe un overflow (un riporto che eccede la capacità di rappresentazione, in questo esempio è un nono bit non gestibile), e rimane il risultato corretto dell'operazione.

Per comprendere meglio il funzionamento vediamo la stessa operazione evidenziando i riporti:

Esempio

```
        11111 111   (riporto)
         0000 1111  (15)
+        1111 1011  (-5)
        ===========
         0000 1010  (10)
```

Gli ultimi due bit a sinistra, ovvero i più significativi, della riga dei riporti esprimono importanti informazioni sulla validità dell'operazione:

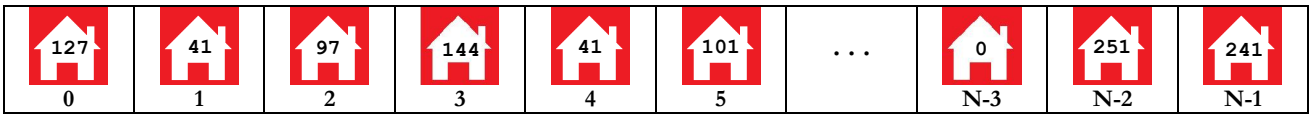
- se i due bit più a sinistra sulla riga dei riporti sono entrambi 0 o 1 allora l'operazione è valida e il numero ottenuto è ben codificato in complemento a due
- se i due bit più a sinistra sulla riga dei riporti sono diversi (uno 0 e l'altro 1) allora l'operazione NON è valida e il numero ottenuto ha generato un overflow

Memorie

Dati e indirizzamento

Ora che abbiamo visto i vari modi di rappresentare informazioni (numeriche, alfabetiche, logiche) possiamo analizzare come sia possibile memorizzare i dati nell'elaboratore elettronico. Iniziamo col dire che non tutte le memorie sono uguali: esistono diversi tipi di memoria, che saranno analizzate in seguito. Per il momento ci soffermiamo su una particolare memoria, detta RAM (Random Access Memory) che costituisce la memoria di lavoro di un elaboratore attivo (acceso) e i cui dati vanno perduti se lo si spegne.

Immaginiamo la RAM come una scheggia di silicio con tante cellette ciascuna delle quali può memorizzare un singolo byte (8 bit). Per comprendere meglio pensiamo alla RAM come una strada con tante casette, tutte uguali. Ogni casetta può ospitare un singolo byte (contenuto o valore) e per poter distinguere le casette occorre un numero, analogo al numero civico delle vie cittadine.

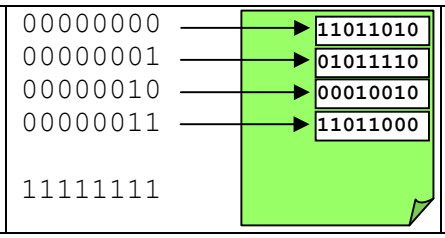


Nel disegno qui sopra ci sono N casette, ciascuna che ospita un singolo byte. Il byte contenuto nella casetta è un dato, ma l'informazione che rappresenta dipende dal contesto, per cui alcuni byte sono numeri, altri sono caratteri e così via.

Appare chiaro allora che ci sono due tipi di numero: il contenuto della casetta e il suo indirizzo (il numero civico). Per esempio nella casetta numero 5 si trova il valore 101.

La RAM funziona all'incirca nello stesso modo. Ogni byte nella memoria ha un suo indirizzo ovvero un numero che lo individua. Nello spazio di memoria può essere contenuto il dato desiderato, sotto forma di zeri e uni.

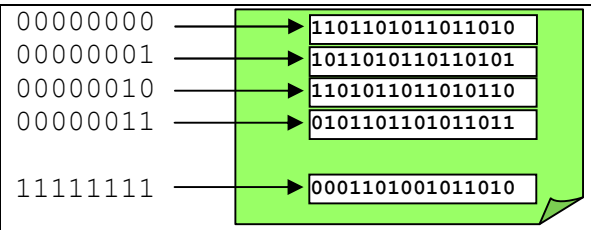
Un indirizzo di RAM è un numero di K bit che individua il singolo elemento memorizzato nella memoria.



Il dato contenuto in memoria è un byte memorizzato nel singolo frammento di memoria.

Poiché l'indirizzo della RAM è formato da K bit appare evidente che K determina la quantità di indirizzi possibili. Per esempio avendo a disposizione soli 8 bit di indirizzamento allora ci sono 2^8 byte in memoria, ovvero 256 byte. In realtà la RAM ha a disposizione molti più bit di indirizzamento. Se per esempio la RAM consente 32 bit di indirizzamento allora può ospitare 2^{32} byte = $2^2 \times 2^{10} \times 2^{10} \times 2^{10}$ = **4 GByte**.

Un altro modo è quello di memorizzare in ogni frammento di memoria NON un singolo byte ma un elemento più grande, per esempio una parola (16 bit); in questo modo ogni indirizzo permette di accedere a un dato di 16 bit.

Un indirizzo di RAM è un numero di K bit che individua il singolo elemento memorizzato nella memoria.		Il dato contenuto in memoria è una parola formata da 16 bit.

In questo secondo caso la stessa quantità di bit di indirizzo della RAM consente una capacità di memoria raddoppiata. Se per esempio la RAM dispone di 32 bit di indirizzamento allora può ospitare $\mathbf{2^{32}}$ word = $\mathbf{2}$ x $\mathbf{2^2}$ x $\mathbf{2^{10}}$ x $\mathbf{2^{10}}$ x $\mathbf{2^{10}}$ **byte** = **8 GByte**.