Viti, dadi e cavetti

Grazie ai cavetti allegati a questo fascicolo potrai realizzare il circuito illustrato a pagina 74. Le viti (di due misure diverse) e i dadi ti serviranno invece per il montaggio della cupola di copertura del robot e della pinza che saranno allegati in seguito: per il momento, dunque, conservali nella scatola portacomponenti. La particolarità dei cavetti che utilizzerai sulla breadboard è il fatto di essere stagnati. L'anima in rame (più fili intrecciati, come nei comuni cavi elettrici) è stata cioè passata in un bagno di stagno che, rivestendola, ha permesso di ottenere un'unica terminazione (invece che più fili), con una precisa sezione, compatibile con la dimensione dei socket della breadboard. In questo modo, risulta più semplice inserire i cavetti nei contatti; inoltre, sono più resistenti all'uso e, adatti a essere usati in molteplici sperimentazioni.

Sopra. **Gli allegati a questo fascicolo: 53 dadi M3, 21 viti lunghe 16 mm, 32 viti lunghe 10 mm e 30 cavetti colorati per la breadboard.**

Le fasi di programmazione

Riprendiamo il discorso sulla gestione dello spazio di memoria riservato alle variabili e alla rappresentazione dei dati in esso contenuti. Abbiamo già detto (a pag. 56) che le variabili possono contenere determinati valori numerici che vengono codificati in modo binario nella memoria; inoltre, sappiamo che lo spazio che una variabile occupa in memoria viene visualizzato dalla Memory Map (a pag. 63). A questo punto, dunque, è opportuno richiamare alcuni concetti legati all'aritmetica e alle possibili rappresentazioni dei numeri.

0	1	2	3	4	5	6	7	8	9

Innanzitutto è importante distinguere tra numero e cifra. Il primo è un ente astratto che rientra in una successione ordinata e che può essere associato al concetto di valore o quantità; la cifra (dall'arabo *sifr*, 'zero') è invece il simbolo o segno grafico atto a rappresentare un numero. I numeri, inoltre, sono infiniti, mentre le cifre sono finite: quelle arabe che usiamo di solito, per esempio sono dieci (0, 1, 2, 3... 9). L'insieme delle regole per cui una certa stringa (serie) di cifre rappresenta un preciso numero o valore costituisce un cosiddetto 'sistema di numerazione'. Così, i numeri con cui siamo abituati a operare quotidianamente sono numeri trascritti nel sistema decimale. La comprensione del valore rappresentato dalla scrittura in cifre di tali numeri ci sembra immediata, eppure dipende da una precisa codifica che, apparentemente ovvia, non è l'unica possibile né la più utile in ogni circostanza. Se, per esempio, scriviamo il numero 46 937, in prima battuta ci sembrerà ovvio che esso rappresenti il valore quarantaseimilanovecentotrentasette; di fatto, invece, tale valore non è immediato ma dipende, invece, da un calcolo preciso (per non confonderla con la lettera *x*, l'operazione di moltiplicazione viene indicata con il carattere *):

$$46\,937 = (7*1)+(3*10)+(9*100)+(6*1000)+(4*10\,000)$$

Quello decimale, infatti è un sistema di numerazione 'posizionale': la stessa cifra assume cioè valori diversi a seconda della sua posizione rispetto alle altre (eventuali) cifre. A partire dalla cifra più a destra (detta 'meno significativa') fino a quella più a sinistra (ossia la 'più significativa'), ciascuna di esse rappresenta rispettivamente le unità, le decine, le centinaia ecc., cioè le potenze di 10. Dunque, l'espressione precedente può essere scritta utilizzando tali potenze:

$$46\,937 = (7*10^0)+(3*10^1)+(9*10^2)+(6*10^3)+(4*10^4)$$

In altre parole, il valore effettivo di un numero decimale si calcola moltiplicando ogni sua cifra, a partire dalla meno significativa, per una potenza crescente di dieci e sommando poi tali prodotti. In questo caso, dunque, il numero 10 viene detto base del sistema di numerazione, perché è il valore di riferimento, in base al quale è possibile generare la rappresentazione di tutti gli altri numeri.

I	V	X	L	C	D	M

Come vedremo tra breve, esistono diversi sistemi di numerazione che, pur essendo posizionali come quello decimale, sono però in base diversa (2, 16 ecc.). D'altra parte, esistono anche sistemi di numerazione non posizionali: un esempio è quello della notazione in numeri romani. La stringa di caratteri numerici romani CCXXIV, ad esempio, rappresenta il numero decimale 224 che si ottiene dalla somma (o sottrazione) delle cifre romane, scritte da sinistra a destra a partire dalla cifra che rappresenta il valore più alto (C=100) fino al più basso (IV=4). In particolare, posto che la cifra C rappresenta il valore 100, X il valore 10, V il valore 5 e I il valore 1, si avrà:

CCXXIV = 100+100+10+10+(5-1)

Come appare evidente, le cifre che ricorrono più di una volta (C e X) rappresentano sempre lo stesso valore (100 o 10), a prescindere dalla loro posizione nella stringa. Nella notazione 224, invece, la cifra 2 che ricorre due volte rappresenta due valori diversi (200 e 20). Questo perché si tratta di un sistema posizionale, mentre quello romano è misto, additivo e sottrattivo. La regola per attribuire il valore corretto a una stringa di cifre romane è che si somma sempre, tranne nei casi in cui l'ordine delle cifre (di solito decrescente) sia crescente: per questo, posto che L rappresenta il valore 50, la stringa LX equivale a 50+10=60 (50>10: l'ordine è decrescente, dunque si somma); invece XL è pari a 50-10=40 (50<10: l'ordine è decrescente, quindi si sottrae).

0	1

Tornando al concetto di sistema di numerazione visto per la base 10, vediamo ora come generalizzarlo: utilizzando basi diverse, infatti, è possibile generare sistemi di numerazione diversi ma 'traducibili' l'uno nell'altro. Una volta scelta una qualsiasi base n, la trascrizione dei numeri potrà contenere cifre comprese tra 0 e n-1. Nel caso del sistema decimale (per cui n=10), ad esempio, le cifre utilizzate per la rappresentazione dei numeri sono tutte quelle comprese tra 0 e 9 (cioè 10-1). La codifica binaria, quella utilizzata in informatica e in elettronica, corrisponde al sistema di numerazione in base 2 (per cui n=2), che prevede una rappresentazione dei numeri tramite la successione di due sole cifre, 0 e 1 (cioè 2-1) che, di fatto, risultano particolarmente adatte a essere 'tradotte' in termini circuitali: solitamente, infatti, si associa 0 ai valori di tensione nulli e 1 ai valori di tensione positivi. Inoltre, applicando la stessa procedura vista prima, cioè moltiplicando ogni cifra (a partire da destra) per le potenze crescenti della base (2), è possibile 'tradurre' un numero scritto nel sistema binario nell'equivalente notazione decimale. Proviamo a vedere perché, ad esempio, il numero binario 11010 ha lo stesso valore del numero decimale 26:

$$11010 = (0*2^0)+(1*2^1)+(0*2^2)+(1*2^3)+(1*2^4) =$$
$$0+2+0+8+16=26$$

Dato invece un numero decimale, la procedura inversa, ossia la determinazione del numero binario corrispondente, è un po' più laboriosa: richiede, infatti, di dividere ripetutamente il numero decimale per la base del sistema binario (2) fino a ottenere 0; la serie dei resti (gli 'avanzi') generati dalle divisioni, letta 'al contrario', costituisce la stringa di cifre del numero binario corrispondente: in pratica, il resto della prima divisione costituisce la cifra meno significativa (cioè, più a destra) di quello binario, mentre l'ultimo resto costituisce la cifra binaria più significativa (cioè, più a sinistra). Per tradurre un numero decimale, per esempio il 26, nell'equivalente notazione binaria, procediamo alla sua divisione per due; quindi, considerando il primo resto come la cifra meno significativa della stringa in base 2, si ottiene il numero binario 11010 (per non confonderla con il segno grafico dei due punti, l'operazione di divisione viene indicata con il carattere /):

Divisione	Risultato	Resto
26/2=	13	0
13/2=	6	1
6/2=	3	0
3/2=	1	1
1/2=	0	1

Tornando ai concetti di numero e cifra, possiamo notare che sistemi di numerazione diversi ricorrono spesso all'uso delle stesse cifre (ad esempio 1 e 0, comuni ai sistemi decimale e binario). Con le due procedure complementari viste prima (la somma delle moltiplicazioni per le potenze crescenti di 10 e la lettura dei resti della serie di divisioni per 2), è possibile passare dalla notazione binaria a quella decimale e viceversa, mantenendo invariato il valore numerico cui le cifre si riferiscono. Se non applicassimo le regole di 'traduzione', invece, la stessa sequenza di cifre rappresenterebbe valori completamente diversi e senza alcun rapporto reciproco: nell'esempio di prima, la sequenza 11010 (uno, uno, zero, uno, zero) è stata considerata scritta nella notazione binaria e, dunque, viene associata al valore 26; se invece fosse stata intesa in base 10, avrebbe avuto un valore ben diverso (undicimiladieci). Dunque, per poter assegnare a una stringa di cifre un valore numerico univoco (uno e solo uno) è necessario specificare a quale sistema di numerazione si sta facendo riferimento. La stringa di cifre 11, di per sé, non significa 'undici'; se però si specifica che va intesa in base 10, allora il suo significato (univoco) è undici, mentre in base 2, ad esempio, significa (univocamente) tre ($1*2^1+1*2^0 = 1+2$). Per specificare la base del sistema di numerazione cui si intende fare riferimento, si adotta la scrittura $(n)_b$ dove n è un numero qualsiasi, b è la base: nell'esempio precedente, dunque, si scriverà rispettivamente $(11)_{10}$ e $(11)_2$, l'uno equivalente a undici, l'altro a tre.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Oltre al sistema binario, in informatica si utilizza spesso la numerazione esadecimale, cioè in base 16 (*hexa*, in greco, significa 'sei'). Dal momento che, come abbiamo detto, le cifre arabe sono solo dieci (da 0 a 9) e che questo sistema ne richiede sedici, per completare il numero mancante di cifre si ricorre all'introduzione delle prime 6 lettere dell'alfabeto (dette cifre alfabetiche). In particolare, si avrà la seguente corrispondenza:

Cifra alfabetica	Valore decimale
A	10
B	11
C	12
D	13
E	14
F	15

La notazione esadecimale 2C6, ad esempio, avrà dunque il seguente valore decimale:

$$2C6 = 6*16^0 + 12*16^1 + 2*16^2 = 6 + 192 + 512 = 710$$

Questo sistema di numerazione può essere utile in informatica perché è 'compatibile' con quello binario, cioè quello in base 2 (16, infatti equivale alla quarta potenza di 2, ossia a 2*2*2*2). Una singola cifra esadecimale può di fatto rappresentare quattro cifre binarie (cioè quattro bit) contemporaneamente e, dunque, permettere una scrittura più compatta di numeri molto grandi che richiederebbero stringhe binarie estremamente lunghe.

Dopo questa digressione sui numeri, possiamo capire meglio l'organizzazione della memoria del microcontrollore del tuo robot e come essa gestisce lo spazio riservato a una variabile. Dichiarare una variabile nel listato di un programma corrisponde, a livello hardware (cioè fisicamente), a occupare nella memoria un blocco di bit (2, 4, 8 o 16) pari alla dimensione dichiarata (bit, nib, byte o word), identificandolo quindi con il nome della variabile dichiarata. Come sappiamo (vedi pag. 56), il numero di bit determina il valore massimo che la variabile può assumere: la regola generale è che un blocco di *n* bit può arrivare a rappresentare numeri positivi interi compresi tra 0 e 2^n-1. Ad esempio, dichiarando una variabile byte (*n*=8 bit), potremo rappresentare tutti i numeri interi compresi nell'intervallo tra 0 e 2^8-1 (cioè 256 – 1 = 255), il che significa, in termini binari, da 00000000 a 11111111. Infatti:

$(00000000)_2 = (0)_{10}$

$(11111111)_2 = 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 1*2^4 + 1*2^5 + 1*2^6 + 1*2^7 =$

$1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = (255)_{10}$

Supponiamo ora di avere il seguente frammento di codice (ossia solo la porzione relativa alla dichiarazione di una variabile e il successivo assegnamento di un dato valore):

```
x    var   nib
...
x = 11
```

Al momento della dichiarazione della variabile (x var nib), nella memoria verrà fisicamente occupato un blocco di 4 bit, identificati con il nome x. Quando poi assegnamo alla variabile x il valore decimale 11 (x=11), il blocco di bit assumerà la configurazione 1011, che è quella che, appunto, corrisponde a tale valore decimale. Applicando la regola di 'traduzione' alla stringa binaria 1011, avremo:

$$1*2^0 + 1*2^1 + 0*2^2 + 1*2^3 = 1 + 2 + 0 + 8 = 11$$

Tuttavia, in PBASIC esiste anche la possibilità di memorizzare direttamente il valore binario di un numero. Vediamo il seguente frammento di codice:

```
x    var   nib
...
x = %1011
```

Con questa scrittura, si assegna alla variabile x (dichiarata esattamente come prima) il numero binario 1011. In particolare, per indicare che si fa riferimento al sistema di numerazione binario, occorre premettere il carattere % alla stringa di cifre in questione (%1011). In maniera analoga, inoltre, è possibile anche utilizzare il sistema esadecimale, premettendo però il carattere $ alla stringa di cifre. Perché l'assegnamento precedente (1011 binario, pari a 11 decimale) sia inteso in termini esadecimali e mantenga lo stesso valore, andrà dunque scritto così (la cifra alfabetica B, come visto prima, equivale al valore decimale 11):

```
x = $B
```

LE FASI DI PROGRAMMAZIONE

Fino a questo punto abbiamo considerato il blocco di bit di una variabile come la rappresentazione binaria di un determinato valore e abbiamo visto come sia possibile assegnare tale valore tramite sistemi di numerazione diversi. Tuttavia il linguaggio PBASIC ti permette di programmare anche considerando singolarmente ogni bit di una variabile. All'interno di un blocco di memoria, infatti, i bit sono numerati da 0 a *n*-1 (*n* è la dimensione della variabile dichiarata e può essere 1, 4, 8, 16) a partire dal bit meno significativo (quello più a destra) fino a quello più significativo (quello più a sinistra). Per accedere ai singoli bit numerati è sufficiente utilizzare la particolare notazione (scrittura) *nomevariabile.bit#*, dove # è un numero intero compreso tra 0 e *n*-1 e indica il numero del singolo bit. Ad esempio, se vogliamo intervenire sul quarto bit di una variabile di nome x e di tipo byte (*n*=8), basterà usare la notazione x.bit4. Questo tipo di scrittura, inoltre, può essere utilizzata sia per leggere il valore del bit sia per modificarlo; scrivendo, ad esempio, l'istruzione x.bit5 = 0, si assegna il valore 0 al quinto bit della variabile x. In questo modo è possibile operare bit per bit (*bit a bit*) sul contenuto di una variabile. Questo tipo di operazione sarà molto utile per programmare la navigazione del robot utilizzando sia i due sensori di contatto (i baffi) sia i due sensori a raggi infrarossi (IR). In particolare, vedremo presto come utilizzare due variabili nib (*n*=4) per memorizzare il valore delle porte relative alle due coppie di sensori (ciascuna memorizzata nei due bit meno significativi della variabile). Includendo in due sole variabili le informazioni rilevate dall'esterno, realizzerai una fusione sensoriale (ottica con una variabile, tattile con l'altra) preziosa per il robot.

Vediamo, infine, un programma d'esempio che riassume tutto quello che abbiamo visto finora. Digita nell'area di editing il listato (sotto, a destra), collega il robot al PC e manda in esecuzione il programma con il pulsante Run della barra degli strumenti. La finestra di debug (sotto) visualizza i risultati dei diversi assegnamenti nel sistema di numerazione (o formato) di volta in volta prescelto e, infine, i risultati dell'assegnamento bit a bit. Vediamo il listato in dettaglio. Innanzitutto si dichiara in nib la variabile x (x var nib); si procede quindi a quattro tipi di assegnamento, ciascuno con valori diversi: decimale (x = 11), binario (x = %1110), esadecimale (x = $C) e bit a bit (x.bit0 = 1… x.bit3 = 1). Dopo ogni assegnamento, le istruzioni debug stampano a video (andando a capo all'occorrenza del comando CR o di ?) il tipo di assegnamento (debug "Assegnamento…") seguito dal valore della variabile nei formati decimale (DEC), binario (BIN) ed esadecimale (HEX). L'ultima porzione di listato ('Assegnamento bit a bit) visualizza anche il valore binario attribuito a ogni singolo bit (debug BIN ? x.bit0, BIN ? x.bit1, BIN ? x.bit2, BIN ? x.bit3).