IMPARIAMO A PROGRAMMARE IN ASSEMBLER

di Valter Di Dio

Prima puntata

Il basic, seppure molto pratico e semplice da apprendere ed utilizzare, non consente, a causa della sua inevitabile lentezza, lo sfruttamento ottimale delle capacità di calcolo di un personal computer. Anche le versioni compilate non arrivano ad aumentare la velocità di esecuzione oltre un fattore dieci nei migliori dei casi.

Alcuni compilatori, in unione a specifici linguaggi simili al basic, riescono ad ottenere risultati migliori, ma sono abbastanza costosi e richiedono, in ogni caso, l'apprendimento di un nuovo linguaggio. A questo punto, chi volesse ottenere dal suo computer gli effetti incredibili di certi programmi "professionali", deve assolutamente cominciare a utilizzare il Linguaggio Macchina.

Nasce così questa serie di articoli sull'Assembler del 6502, il microprocessore che "manda avanti" l'Apple, il Vic e molti altri personal fra i più diffusi. Programmare in Linguaggio Macchina non è così difficile come potrebbe sembrare guardando i programmi altrui; come tutte le cose, preso per il giusto verso e con gli "attrezzi" adeguati, non è più complicato dell'uso di una comune calcolatrice programmabile. Una cosa va precisata: questi sono articoli introduttivi, e quindi destinati a tutti coloro che vorrebbero imparare ad usare l'Assembler ma hanno paura di affrontare un testo specifico che darebbe per scontate troppe cose. Proprio per questo ci soffermeremo anche sulle cose più ovvie, che forse per qualcuno potrebbero non esserlo tanto.

Se, nonostante ciò, ci fossero ancora dei problemi, non esitate a scriverci.

Fateci sapere le vostre preferenze per quello che riguarda gli impieghi del linguaggio macchina ed, eventualmente, le necessità particolari. Fondamentalmente siamo contrari ai "compiti a casa" ma vi consigliamo vivamente di mettere in pratica il più possibile quanto leggete facendo qualche prova, magari banale, per vedere a che punto siete arrivati. Le prime volte vi capiterà di inchiodare la macchina o riempire lo schermo di mostri strani, ma prima o poi comincerete a realizzare programmi che "girano" e potrete finalmente appuntarvi sul petto il distintivo dei "veri" programmatori.

Introduzione

L'insieme delle istruzioni direttamente eseguibile da un calcolatore e le regole sintattiche che ne determinano la validità costituiscono il Linguaggio Base o Linguaggio Macchina dell'elaboratore. La sua circuiteria interna è stata realizzata in modo da comprendere esclusivamente tali istruzioni. Qualunque altro linguaggio, ideato al solo scopo di facilitare il lavoro dei programmatori, deve, prima di poter essere eseguito, venir tradotto nel linguaggio base della specifica macchina su cui dovrà operare.

Prima dell'avvento dei microprocessori, ogni macchina, anche modelli diversi di una stessa casa, utilizzava un set di istruzioni differente ed era impensabile il trasporto di un programma in linguaggio macchina da un elaboratore ad un altro.

Oggi, invece, è possibile trasferire programmi in linguaggio macchina tra tutte le macchine che usano lo stesso microprocessore cambiando solo le routine di entrata/uscita.

Un tipico esempio è dato dal CP/M per le macchine basate sullo Z80.

A volte si fa confusione tra linguaggio macchina e Assembler. Il linguaggio macchina è un programma assoluto, ossia che usa come istruzioni dei codici binari; l'Assembler, per contro, fa parte dei linguaggi simbolici dal momento che usa dei simboli (caratteri ASCII) per codificare le proprie istruzioni.

Anche il Basic è un linguaggio simbolico ma, a differenza dell'Assembler, ha perso il rapporto uno a uno tra istruzioni simboliche e codici macchina permettendo di sganciarsi dall'uso diretto della memoria e dei registri di calcolo.

La programmazione in Assembler è, dal punto di vista logico, la stessa cosa del linguaggio macchina; l'Assembler, infatti, non fa altro che tradurre una serie di codici, detti mnemonici, nei corrispondenti binari; consente di definire delle locazioni di memoria o dei punti di programma (il che in fondo è la stessa cosa) con delle etichette alfanumeriche; è provvisto inoltre di editing e permette di ottenere dei programmi RILOCABILI, che possono essere caricati in qualsiasi parte della memoria da un adatto programma di caricamento.

Esistono varie versioni di Assembler; dal più semplice, un blocco notes, una penna e tanta pazienza, al Miniassembler, che si trova nell'interprete Integer Basic e che useremo per i nostri esempi, ai più complessi e potenti tipo il LISA. Anche se, per gli esempi e per quello che riguarda le routine del monitor, ci riferiremo al sistema Apple II, tutto quello che riguarda le microistruzioni e la logica generale resta valido per qualsiasi macchina che usi come CPU il 6502 (Vic 20, Commodore 64, Atari, e altri).

Dal momento che ci accostiamo al linguaggio macchina si presume una normale conoscenza del basic e una certa dimestichezza con la programmazione in generale.

L'intenzione originale era di cominciare

Tabella 1

Binario	Dec.	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

subito ad accendere l'Apple e vedere qualche esempio pratico, ma, dal momento che questi articoli sono destinati a chi ancora nulla conosce della aritmetica binaria o dei numeri esadecimali, dovremo necessariamente dedicare questa prima puntata all'algebra binaria.

Chi si ritiene sufficientemente esperto può saltare questa parte, ma pensiamo comunque che una ripassata non faccia certo male.

Il dato binario

In una cella di memoria a otto bit può essere contenuto un numero che va da 0 a $11111111_{(2)} = 255_{(10)}$. Per convertire un numero binario in esadecimale è sufficiente separare i bit a quattro a quattro partendo da destra e tradurre ciascuna "quartina" nella corrispondente cifra esadecimale: ad esempio 10110011 si divide in 1011 che, come si vede dalla tabella 1, corrisponde a $B e 0011 che vale $3; il nostro numero equivale perciò a $B3. Il segno del dollaro davanti al numero indica che questo è in base 16 così come nell'uso comune dell'Apple.

Per sapere ora quanto vale il nostro numero esadecimale nel più familiare sistema decimale si possono usare tre metodi.

Il primo, molto rapido, consiste nell'uso delle apposite tavole di conversione (vedi MC n. 19).

Si può anche calcolare direttamente il valore decimale moltiplicando ciascuna cifra del numero esadecimale per il peso della sua posizione: 1 per la prima, 16 per la seconda, 16^2 per la terza e così via; questo sistema viene usato di solito nei programmi.

Il metodo più comodo di convertire un esadecimale in decimale resta comunque il passaggio per l'equivalente binario.

Per passare da esadecimale a binario si segue il procedimento inverso a quello, spiegato prima, per la conversione da binario a esadecimale; ossia, molto semplicemente, si scrive ciascuna cifra direttamente in binario. Il numero binario può ora essere facilmente trasformato in decimale con un metodo molto veloce e "divertente": il DOUBLE-DABBLE (raddoppia e somma).

Vediamo come si sviluppa.

Si inizia col raddoppio (Double) della cifra binaria più significativa (quella a sinistra) e si addiziona al valore ottenuto la cifra che si trova alla immediata destra (Dabble). Quindi si raddoppia il risultato parziale e si aggiunge 1 o 0 a seconda della cifra che segue. Si va avanti così fino all'ultima cifra del numero. Attenzione a non raddoppiare ancora, l'ultima operazione deve essere una somma! Provate, eventualmente facendo riferimento alla Tabella 2.

Con un po' di pratica si può eseguire il conto anche a mente.

Adesso che sappiamo fare le conversioni proviamo a eseguire qualche operazione aritmetica in binario:

```
00000101 ($05) +
00001110 ($0E) =
----------------
00010011 ($13)
```

Se a qualcuno viene diverso riprovi confrontando con la tabella 3, in cui C-1 rappresenta il riporto della somma precedente, C il riporto attuale ed S il risultato della somma di X1, X2 e C-1.

Fin qui tutto semplice, ma se invece di sommare avessimo sottratto? Il risultato sarebbe stato negativo e, mentre sul foglio di carta basta mettere un trattino davanti a un numero per dire che è negativo, nella memoria di un computer non è prevista la possibilità di mettere trattini!

Si ricorre, allora, ad un piccolo trucco: se è vero che in una cella di memoria possiamo avere 256 combinazioni diverse, ci basta spostare in avanti l'origine per avere 128 numeri negativi e 127 numeri positivi.

Dal momento che 7 bit sono sufficienti a contare fino a 127 possiamo usare il bit che avanza (per convenzione il più significativo) come bit di segno; esso varrà "1" se esiste segno negativo o "0" se il dato è positivo.

A questo punto qualcuno potrebbe obiettare che così facendo vengono fuori due zeri: uno positivo uguale a 00000000 e uno negativo (?) che corrisponde a 10000000.

Esiste anche un secondo problema, ben più grave, che a prima vista non si nota ma che si risolve contemporaneamente al precedente. Se noi proviamo a sommare, con le normali regole dell'algebra binaria, un numero negativo, per esempio -5 che corrisponde a 10000101, e un numero positivo, prendiamo 7 = 00000111, otteniamo:

```
(+7) 00000111 +
(—5) 10000101 =
----------------
10001100
```

ovvero -12; risultato naturalmente errato. Questo dimostra che la convenzione usata fin qui per i numeri negativi non è sufficiente, da sola, per poter oltre che rappresentare anche gestire un'algebra negativa.

La soluzione del problema è molto semplice: basta, per convenzione (un'altra!), usare per i numeri negativi, al posto del numero con segno prima definito, il suo complemento a due. Per complemento a due si intende il complemento a uno, ossia lo scambio degli "1" con "0" e degli zeri con "1", e successivamente l'incremento del risultato di uno. Allora, il complemento a due di (+5) = 00000101, è uguale al suo complemento a uno, cioè 11111010, più uno, quindi 11111011.

Riproviamo ora la somma:

```
(+3) 00000011 +
(—5) 11111011 =
----------------
(-2) 11111110
```

questo risultato è corretto; si è visto inoltre che si può operare sul bit di segno con le normali regole e ottenere automaticamente il segno corretto.

Riassumendo si passa da un numero positivo al suo equivalente negativo sempli-

Tabella 2

```
1     0     1     1     1

1×2=2
      +
      0
      =
      2×2=4
            +
            1
            =
            5×2=10
                  +
                  1
                  =
                  11×2=22
                        +
                        1
                        =
                        23
```

Esempio di Double/Dabble

Tabella 3

X1	X2	C-1	S	C
0	0	0	0	0
0	0	1	1	0
1	0	0	1	0
1	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	1	0	0	1
1	1	1	1	1

cemente eseguendo il complemento a due di tutto il numero, segno compreso, e viceversa.

Usando il complemento a due per i numeri negativi non esiste più lo "zero negativo" che viene sostituito da -128, si spiega così perché i numeri negativi sono sempre uno di più dei positivi.

La tabella 4 mostra alcuni numeri negativi in complemento a due. Se non avete capito bene, vi consigliamo di provare a fare un po' di esercizi. Se avete capito bene, ve lo consigliamo lo stesso.

Dal momento che non è pensabile avere un campo numerico che vada solo da 127 a -128, si usa riunire più byte (solitamente 2) per ottenere una estensione maggiore.

Il dato BCD

Un altro metodo usato per codificare dei dati numerici all'interno della memoria di un elaboratore è il BCD: Binary Coded Decimal. Consiste nel tradurre in binario non il numero intero ma le singole cifre decimali che lo compongono. Per esempio, 2345 diventa 0010, 0011, 0100, 0101.

Si nota subito che, per poter codificare ciascuna cifra decimale, occorrono quattro bit (3 non bastano dal momento che 111 = 7), i quali potrebbero contare fino a sedici. Questo significa che si sprecano sei combinazioni per ogni cifra. In un byte si possono impaccare due cifre BCD, quindi il massimo numero che può contenere una cella di memoria passa da 255 a soli 99; inoltre, ora, occorre gestire il riporto tra la prima e la seconda cifra di ciascun byte e correggere i risultati di tutte le operazioni binarie che contengano qualcuno dei sei codici non usati dal BCD. Per fortuna il 6502 è provvisto di istruzioni molto potenti per la gestione diretta dell'aritmetica BCD.

Il BCD è usato soprattutto in contabilità dal momento che permette di conservare tutte le cifre significative di un numero, a scapito però del fatto che il formato dei numeri non è costante e complica la gestione di vettori e matrici.

+	Codice in complemento a 2	−	Codice in complemento a 2
+127	01111111	−128	10000000
+126	01111110	−127	10000001
+125	01111101	−126	10000010
...		−125	10000011
		...	
+ 65	01000001	− 65	10111111
+ 64	01000000	− 64	11000000
+ 63	00111111	− 63	11000001
...		...	
+ 33	00100001	− 33	11011111
+ 32	00100000	− 32	11100000
+ 31	00011111	− 31	11100001
...		...	
+ 17	00010001	− 17	11101111
+ 16	00010000	− 16	11110000
+ 15	00001111	− 15	11110001
+ 14	00001110	− 14	11110010
+ 13	00001101	− 13	11110011
+ 12	00001100	− 12	11110100
+ 11	00001011	− 11	11110101
+ 10	00001010	− 10	11110110
+ 9	00001001	− 9	11110111
+ 8	00001000	− 8	11111000
+ 7	00000111	− 7	11111001
+ 6	00000110	− 6	11111010
+ 5	00000101	− 5	11111011
+ 4	00000100	− 4	11111100
+ 3	00000011	− 3	11111101
+ 2	00000010	− 2	11111110
+ 1	00000001	− 1	11111111
+ 0	00000000		

Tabella 4

Il Float

Il metodo usato dall'Applesoft per gestire un numero reale è invece il FLOAT dove ogni numero viene trasformato in una mantissa di quattro byte con segno e un esponente di un byte sempre con segno.

In questo modo il formato dei numeri è sempre di cinque byte, qualsiasi sia la dimensione del dato, a scapito questa volta della precisione, dal momento che le cifre della mantissa sono solo nove. Occorre inoltre un apposito programma in linguaggio macchina per poter gestire questa aritmetica.

Conclusione

Obiettivamente, a livello di programmazione in assembler non avete ancora imparato nulla. Ripetiamo, però, che abbiamo ritenuto di dover desistere dalla nostra idea originaria, di "imparare facendo" ponendosi direttamente davanti alla macchina accesa.

Quello che abbiamo detto in questa prima puntata ci è sembrato una specie di indispensabile premessa; nel prossimo numero potremo comunque affrontare l'argomento con un approccio più concreto, cioè senza lasciare spento il computer.

Nel frattempo, vi consigliamo di "giocare un po'" con i numeri e le conversioni: non serve a niente da un punto di vista fine a sé stesso, ma sarà utile, in futuro, aver acquisito una certa dimestichezza con questi problemi. MC

HARDWARE + SERVIZIO

Il nostro concetto di valore

HONEYWELL:
Stampanti ad aghi con matrice 9×9 seriali o parallele con velocità di stampa da 100 fino a 400 caratteri per secondo, da 80 a 220 colonne, con percorso bidirezionale ottimizzato e la più completa gamma di caratteri, grafica inclusa.
Honeywell; le printers tutte Italiane a conferma di una immagine Made in Italy, che sempre più si impone sui mercati Internazionali.
SERVIZIO
DATA BASE OEM-D è il distributore di prodotti OEM che vi offre soprattutto un servizio di prim'ordine.
I nostri tecnici vi assicurano la massima collaborazione durante l'interfacciamento delle periferiche con il vostro sistema.
L'assistenza tecnica e la manutenzione – tra le più importanti performances della DATA BASE OEM-D – vi garantiscono la costante efficenza dei nostri prodotti.
DATA BASE OEM-D significa qualità e servizio.
DATA BASE OEM-D è sicurezza.

DB O.E.M.-D

VIMERCATE (MI) Via Banfi, 19 Tel.039/664581/2/3 • PADOVA - Via Trasea, 2 Tel. 049-654463 • SASSUOLO (MO) - P.zza Amendola, 1 Tel.0536-802562• ROMA - Via A. Leonori, 36 Tel. 06/5420305-5423716 • ROMA - Via Dell'Oceano Atlantico, 226/228 Tel. 06/5921191- 5921136-5911010 • TORINO - Via Avigliana, 2 bis Tel. 011/747112-745356 • POZZUOLI - NAPOLI - Via Righi, 8 tel. 081/7601939-7603429-7603633