# IMPARIAMO A PROGRAMMARE IN ASSEMBLER

**Seconda parte**

di Valter Di Dio

*Eccoci alla seconda puntata di questi articoli sul linguaggio macchina del 6502, il microprocessore dell'Apple II e III, del Vic 20, del Commodore 64, dell'Atari e tanti altri diffusissimi personal.*

*Come anticipato in conclusione nello scorso articolo, cominceremo subito ad usare la macchina per fare qualche prova in linguaggio macchina; vedremo soprattutto come si usa il Monitor per scrivere, listare ed eseguire dei programmi in linguaggio macchina.*

Un programma in linguaggio macchina consiste in una serie, generalmente contigua e successiva, di codici istruzione (valori esadecimali che vengono interpretati dal microprocessore come comandi) e di dati (numeri esadecimali anche questi), che si trovano nella memoria del computer. Immaginate di avere un block notes e di scrivere degli ordini per un vostro dipendente assolutamente cretino! Quando gli direte VA! lui inizierà dalla prima pagina del blocco ed eseguirà esattamente quello che ci troverà scritto poi passerà, salvo diverse indicazioni scritte sul precedente foglio, alla pagina successiva e così via.

Se immaginate ora che il dipendente oltre a essere cretino sia pure analfabeta ed abbia imparato solo a leggere i numeri e ad associare a certi numeri determinate azioni, avrete un'idea abbastanza fedele del microprocessore. Comprenderete quindi perché occorra molta attenzione a quello che si scrive sul notes. Perché se è possibile barrare una pagina non è assolutamente fattibile l'inserimento di foglietti tra le pagine ed ogni aggiunta al programma ne comporta l'intera riscrittura.

Per scrivere un programma in linguaggio macchina non resta quindi che conoscere i codici dei comandi e scriverli nell'ordine desiderato in successive celle di memoria, poi diremo al microprocessore da quale locazione iniziare a leggere ed eseguire i comandi.

Prima di iniziare a scrivere un programma vediamo come è possibile leggere o modificare il contenuto delle celle di memoria del nostro Apple tramite il Monitor. Per i possessori di altri personal, ad esempio il Vic 20, è necessario innanzitutto la disponibilità del Monitor, che per il Vic si chiama VIC-MON, e quindi guardare sul manuale gli appositi comandi; inoltre le zone di RAM libere per i programmi e le mappe RAM del video in testo o grafico differiscono da macchina a macchina e addirittura dipendono da che tipo di espansione sia montata, tutte queste informazioni si trovano comunque sui manuali e una volta note non è difficile adattare quello che diremo alla vostra macchina. Per il Vic molte di queste informazioni si trovano su Vic Revealed: vedi recensione su MC n. 15.

Accendete l'Apple, date un NEW e battete ora:

POKE 1010,105
POKE 1011,255
CALL - 1169

Così facendo abbiamo spostato il puntatore del RESET in modo da non ritornare al basic se, per fermare un programma in linguaggio macchina andato in loop, fossimo costretti a premere il tasto di RESET.

Ora premete il RESET o battete il famoso CALL - 151. Comparirà un asterisco seguito dal cursore lampeggiante; è il PROMPT che ci informa dell'avvenuto passaggio al Monitor e che l'Apple è pronto ad accettare un nostro input.

Da questo momento siamo padroni assoluti della macchina e ogni nostro comando verrà eseguito senza "discussioni". Facciamo un esempio: immaginate di essere entrati con un vostro amico che conosce il cinese in un ristorante di Shanghai dove avete intenzione di fare la prima colazione. Se dite al vostro amico di ordinarvi: «Un cappuccino e due bacarozzi con la panna» il vostro amico vi avvertirà dell'errore e non passerà l'ordinazione al cameriere; ma se, disponendo di un rudimentale vocabolario italiano/cinese, aveste abbandonato il vostro interprete e vi foste rivolti direttamente al cameriere avreste probabilmente ottenuto i tanto sospirati "bacarozzi con la panna".

Dal momento che si passa al Monitor siamo in queste precise condizioni. Il monitor tenterà sempre di eseguire il comando ricevuto ed essendo, in genere, composti da un solo carattere, la cosa sarà spesso possibile. Se ad esempio al posto di 300L (comando che disassembla 20 istruzioni a partire da 300) aveste scritto 300LIST il monitor eseguirebbe il listato voluto (L) poi l'inverse video (I) quindi i comandi (S) step e (T) trace che però non sono attivi negli Apple con la ROM Autostart.

A questo punto, onde evitare spiacevoli conseguenze, è consigliabile togliere dal Driver il floppy-disk; infatti il DOS (il programma che gestisce la scrittura e la lettura da disco) è un programma in linguaggio macchina e le sue routine possono essere tranquillamente lanciate con i comandi del monitor. Niente di più facile quindi che un comando errato finisca per sporcare irrimediabilmente il dischetto che si trova nel Drive.

## I comandi del monitor

Si ottiene la lettura di una locazione di memoria scrivendo semplicemente il numero della locazione (in esadecimale) e battendo «return», es.

FF3A «return» cui verrà risposto FF3A - A9

per leggere le locazioni successive basterà dare solo «return» e verranno stampati i contenuti delle locazioni successive fino alla prima che termini per 8 o per 0, oppure si può scrivere la locazione finale separata da un punto da quella iniziale, es.:

*FB2F.FB5C «return»

```
FB2F- A9
FB30- 00 85 48 AD 56 C0 AD 54
FB38- C0 AD 51 C0 A9 00 F0 0B
FB40- AD 50 C0 AD 53 C0 20 36
FB48- F8 A9 14 85 22 A9 00 85
FB50- 20 A9 28 85 21 A9 18 85
FB58- 23 A9 17 85 25
*
```

## Scrittura

Per depositare un numero in una locazione è sufficiente scrivere «locazione» : «valore». Ricordate che mentre la locazione può avere fino a quattro cifre HEX il contenuto si ferma a due, dato che 255 = $FF è il massimo valore che possiamo mettere in ciascuna cella di memoria.

Proviamo subito

*300 «return»
0300 - 02
*300:A9
*300 «return»
0300 - A9

In caso di locazioni successive non è necessario ripetere ogni volta l'indirizzo ma è possibile scrivere solo i dati separandoli con uno spazio; se, nel caso di linee molto lunghe, premete il return, per continuare basta battere subito dopo l'asterisco del Prompt i duepunti e proseguire con i dati.

ATTENZIONE: non toccate (almeno per ora) le locazioni che vanno da 0000 a 02FF.

Per quelli che lo faranno lo stesso e cui non accadrà nulla vorrà dire che sono stati fortunati, per tutti gli altri che si ritroveranno con l'Apple bloccato o lo schermo pieno di "alieni" diciamo subito che la cosa migliore da fare è quella di spegnere l'Apple e poi, dopo alcuni secondi, riaccenderlo (ricordate di reinserire il dischetto).

Se provate a cambiare il valore di locazioni maggiori di D000 non otterrete alcun effetto dal momento che in quella zona è situata la ROM (che per definizione non si cancella); se invece provate a toccare le celle che vanno da C000 a CFFF il risultato può a volte essere strano dato che in questa zona si trovano i Soft-switch delle pagine grafiche, le entrate/uscite del registratore a cassette, delle paddle e l'altoparlante (C030) e, da C100 in poi c'è lo spazio riservato alle schede di espansione (compreso il controller dei dischi).

Vediamo ora un comando molto utile che verrà usato spessissimo.

## Il List

Come già avrete immaginato questo comando consente di listare (proprio come in basic) un programma in linguaggio macchina. Il comando è formato dalla semplice lettera L. Una cosa importante distingue un LIST del Basic dall'equivalente del Monitor. Il list del monitor vuole che si specifichi da che punto della memoria debba cominciare la sua prestazione. Da notare che in linguaggio macchina sia le istruzioni che i dati hanno lo stesso formato: sono infatti dei semplici numeri, per cui $A9 è sia una istruzione che il numero 169! Anche se questo sembra generare una certa confusione in realtà il problema non è poi così grave; infatti il microprocessore è realizzato in modo da interpretare il primo valore che incontra come ISTRUZIONE.

Vediamo un esempio pratico

*FB2FL «return»

riportato nel riquadro a fianco.

```
FB2F-  A9 00     LDA  #$00
FB31-  85 48     STA  $48
FB33-  AD 56 C0  LDA  $C056
FB36-  AD 54 C0  LDA  $C054
FB39-  AD 51 C0  LDA  $C051
FB3C-  A9 00     LDA  #$00
FB3E-  F0 0B     BEQ  $FB4B
FB40-  AD 50 C0  LDA  $C050
FB43-  AD 53 C0  LDA  $C053
FB46-  20 36 FB  JSR  $FB36
FB49-  A9 14     LDA  #$14
FB4B-  85 22     STA  $22
FB4D-  A9 00     LDA  #$00
FB4F-  85 20     STA  $20
FB51-  A9 28     LDA  #$28
FB53-  85 21     STA  $21
```

| INDIRIZZAMENTI | IMPLICITO | IMMEDIATO | ASSOLUTO | ZERO PAGE | ZERO PAGE ,X | ZERO PAGE ,Y | ASSOLUTO ,X | ASSOLUTO ,Y | INDICIZZATO INDIRETTO | INDIRETTO INDICIZZATO | RELATIVO | INDIRETTO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DISASSEMBLATO | = | #$ n | $ n n | $ n | $ n,X | $ n,Y | $ n n,X | $ n n,Y | $ (n,X) | $ (n),Y | $ n | $(n) |
| CODICE LUNGH. | 1 | 2 | 3 | 2 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | 3 |
| ADC | | 69 | 6D | 65 | 75 | | 7D | 79 | 61 | 71 | | |
| AND | | 29 | 2D | 25 | 35 | | 3D | 39 | 21 | 31 | | |
| ASL | 0A | | 0E | 06 | 16 | | 1E | | | | | |
| BCC | | | | | | | | | | | 90 | |
| BCS | | | | | | | | | | | B0 | |
| BEQ | | | | | | | | | | | F0 | |
| BIT | | | 2C | 24 | | | | | | | | |
| BMI | | | | | | | | | | | 30 | |
| BNE | | | | | | | | | | | D0 | |
| BPL | | | | | | | | | | | 10 | |
| BRK | 00 | | | | | | | | | | | |
| BVC | | | | | | | | | | | 50 | |
| BVS | | | | | | | | | | | 70 | |
| CLC | 18 | | | | | | | | | | | |
| CLD | D8 | | | | | | | | | | | |
| CLI | 58 | | | | | | | | | | | |
| CLV | B8 | | | | | | | | | | | |
| CMP | | C9 | CD | C5 | D5 | | DD | D9 | C1 | D1 | | |
| CPX | | E0 | EC | E4 | | | | | | | | |
| CPY | | C0 | CC | C4 | | | | | | | | |
| DEC | | | CE | C6 | D6 | | DE | | | | | |
| DEX | CA | | | | | | | | | | | |
| DEY | 88 | | | | | | | | | | | |
| EOR | | 49 | 4D | 45 | 55 | | 5D | 59 | 41 | 51 | | |
| INC | | | EE | E6 | F6 | | FE | | | | | |
| INX | E8 | | | | | | | | | | | |
| INY | C8 | | | | | | | | | | | |
| JMP | | | 4C | | | | | | | | | 6C |
| JSR | | | 20 | | | | | | | | | |
| LDA | | A9 | AD | A5 | B5 | | BD | B9 | A1 | B1 | | |
| LDX | | A2 | AE | A6 | | B6 | | BE | | | | |
| LDY | | A0 | AC | A4 | B4 | | BC | | | | | |
| LSR | 4A | | 4E | 46 | 56 | | 5E | | | | | |
| NOP | EA | | | | | | | | | | | |
| ORA | | 09 | 0D | 05 | 15 | | 1D | 19 | 01 | 11 | | |
| PHA | 48 | | | | | | | | | | | |
| PHP | 06 | | | | | | | | | | | |
| PLA | 68 | | | | | | | | | | | |
| PLP | 28 | | | | | | | | | | | |
| ROL | 2A | | 2E | 26 | 36 | | 3E | | | | | |
| ROR | 6A | | 6E | 66 | 76 | | 7E | | | | | |
| RTI | 40 | | | | | | | | | | | |
| RTS | 60 | | | | | | | | | | | |
| SBC | | E9 | ED | E5 | F5 | | FD | F9 | E1 | F1 | | |
| SEC | 38 | | | | | | | | | | | |
| SED | F8 | | | | | | | | | | | |
| SEI | 78 | | | | | | | | | | | |
| STA | | | 8D | 85 | 95 | | 9D | 99 | 81 | 91 | | |
| STX | | | 8E | 86 | | 96 | | | | | | |
| STY | | | 8C | 84 | 94 | | | | | | | |
| TAX | AA | | | | | | | | | | | |
| TAY | A8 | | | | | | | | | | | |
| TSX | BA | | | | | | | | | | | |
| TXA | 8A | | | | | | | | | | | |
| TXS | 9A | | | | | | | | | | | |
| TYA | 98 | | | | | | | | | | | |

Tabella 1

**Significato dei codici mnemonici del 6502**

**ADC** Somma con Carry
**AND** Effettua l'And tra A e la Memoria
**ASL** Scorrimento a Sinistra di un Bit
**BCC** Diramazione se il Carry è vuoto
**BCS** Diramazione se il Carry è pieno
**BEQ** Salto per Zero
**BIT** Testa i bit in memoria
**BMI** Salta per risultato Negativo
**BNE** Salta per Diverso da Zero
**BPL** Salta se Positivo
**BRK** Break
**BVC** Salta se non c'è Overflow
**BVS** Salta in caso di Overflow
**CLC** Pulisce il Carry
**CLD** Toglie il Decimal Mode
**CLI** Pulisce il flag di Interrupt
**CLV** Pulisce il flag di Overflow
**CMP** Confronta la Memoria con A
**CPX** Confronta la Memoria con X
**CPY** Confronta la Memoria con Y
**DEC** Decrementa la memoria
**DEX** Decrementa X
**DEY** Decrementa Y
**EOR** Or Esclusivo tra A e M
**INC** Incrementa M di uno
**INX** Incrementa X di uno
**INY** Incrementa Y di uno
**JMP** Salta a
**JSR** Salta a Subroutine
**LDA** Immagazzina in A
**LDX** Immagazzina in X
**LDY** Immagazzina in Y
**LSR** Scorrimento a destra di un Bit
**NOP** Operazione nulla
**ORA** Or tra A e M
**PHA** Spinge A sullo Stack
**PHP** Spinge P sullo Stack
**PLA** Riprende A dallo Stack
**PLP** Riprende P dallo Stack
**ROL** Rotazione a sinistra di un bit
**ROR** Rotazione a destra
**RTI** Ritorno da Interrupt
**RTS** Ritorno da Subroutine
**SBC** Sottrae con Carry
**SEC** Setta il Carry
**SED** Predispone il Decimal Mode
**SEI** Setta il flag di Interrupt
**STA** Immagazzina A in Memoria
**STX** Immagazzina X in M
**STY** Immagazzina Y in M
**TAX** Trasferisce A in X
**TAY** Trasferisce A in Y
**TSX** Trasferisce il puntatore dello Stack in X
**TXA** Trasferisce X in A
**TXS** Trasferisce X nel puntatore dello Stack
**TYA** Trasferisce Y in A

Il listato è composto da tre sezioni: la prima a sinistra contiene l'indirizzo di partenza dell'istruzione, seguono poi i codici esadecimali che sono stati realmente inseriti in memoria, quindi il codice mnemonico seguito, eventualmente, dai dati o da un indirizzo.

Il primo numero incontrato $A9 è stato interpretato come l'istruzione LDA # (vedremo in seguito il significato esatto) seguita dal dato di un solo byte $00 (il dollaro che precede il valore significa, al solito, che questo è esadecimale).

Il numero che segue deve quindi essere una nuova istruzione essendo terminata quella precedente; anche questa è lunga due Byte. La successiva è invece di tre byte e così via. Una tabella interna informa il microprocessore di quanto deve incrementare il contatore di programma, prima colonna a sinistra nel listato, a seconda del tipo di istruzione che sta eseguendo. È chiaro che se avessimo fatto iniziare il listato da un punto a caso ne sarebbe risultato un listato incomprensibile. Provate infatti:

```
*FB4FL

FB4F-   85 20        STA   $20
FB51-   A9 28        LDA   #$28
FB53-   85 21        STA   $21
FB55-   A9 18        LDA   #$18
```

e confrontate le prime tre istruzioni con quelle ottenute da:

```
*FB50L

FB50-   20 A9 28     JSR   $28A9
FB53-   85 21        STA   $21
FB55-   A9 18        LDA   #$18
FB57-   85 23        STA   $23
```

Notate che ci siamo spostati solo di una locazione in avanti!

Per proseguire un listato basta battere semplicemente L «return», è anche possibile battere più di una L se si desidera un listato più lungo. Verranno listate venti istruzioni per ciascuna L senza interruzioni.

Ultimo comando che esaminiamo è quello che ci permette di lanciare in esecuzione un programma in linguaggio macchina. Equivale al RUN del Basic solo che, come la List, richiede il punto di inizio del programma. Ce ne serviamo subito per ricollegare il DOS che la pressione del tasto di RESET lascia escluso. Se infatti provate a dare un comando DOS (es. CATALOG) il Monitor, non conoscendo il comando C, vi risponde con un bip e il nuovo PROMPT. Battete allora: *3EAG «return» e il DOS sarà ricollegato. Il comando G (in inglese GO) è appunto il RUN; in realtà corrisponde più a un GOSUB dal momento che non esiste in linguaggio macchina l'istruzione END e tutte le routine e i programmi terminano con un RTS che significa Return from Subroutine.

In figura 1 trovate una serie di subroutine del Monitor che possono essere lanciate facilmente dal comando G. Ad esempio FC58G effettua la pulizia dello schermo (HOME).

Adesso che siamo in grado di scrivere, listare e mandare in esecuzione un programma possiamo iniziare il discorso sulle istruzioni del 6502.

## Le istruzioni e i registri

Esattamente come per il Basic si dividono in istruzioni di assegnazione, di calcolo o confronto e di salto. Andiamo per ordine: le istruzioni di assegnazione sono quelle che permettono di depositare in una certa locazione un valore qualsiasi e, possibilmente, anche di andarlo a riprendere; in Basic X = 12 è una istruzione di assegnazione ed anche A = X. A questo punto viene fuori il problema delle variabili. In Basic ne esistono vari tipi che possiamo chiamare con i nomi più disparati, in assembler esistono solo due tipi di variabile: le locazioni di memoria, che sono abbastanza numerose, e i Registri interni del microprocessore che sono solo tre (in realtà ce ne sono altri tre di cui uno a sedici bit ma non possono essere usati come variabili). Una enorme differenza distingue questi due tipi di variabile: nelle locazioni di memoria non è possibile eseguire alcun tipo di calcolo o altra operazione logica, non è inoltre possibile la scrittura o la lettura immediata di una locazione di memoria, non posso cioè dire al microprocessore di scrivere direttamente $F2 nella locazione 300. Le locazioni della RAM vanno quindi viste un po' come la memoria delle prime calcolatrici tascabili in cui era possibile solo depositare un risultato intermedio (trasferendovelo dal visualizzatore) e riprenderlo poi al momento opportuno (rimettendolo nuovamente nel visualizzatore). Nel caso di richiamo della memoria il contenuto del visualizzatore veniva naturalmente perso, mentre la scrittura in memoria non modificava il dato del visualizzatore. È chiaro che per fare delle operazioni doveva esistere almeno un secondo registro che non fosse il visualizzatore o la memoria, questo registro veniva chiamato X. Era perciò possibile sommare il contenuto del visualizzatore col contenuto del registro X o con il contenuto della memoria; il risultato ovviamente veniva rimesso nel visualizzatore che prendeva così il nome di Accumulatore, in quanto accumulava uno degli operandi durante il calcolo e il risultato alla fine. Nelle operazioni più complesse, ad esempio una divisione, due registri non erano più sufficienti, il terzo registro venne chiamato, con molta fantasia, Y. In alcune calcolatrici esiste tuttora un tasto che consente di scambiare il contenuto dei registri X e Y. Come avrete capito l'Accumulatore, X e Y sono appunto i tre registri interni del 6502 e il loro funzionamento è molto simile a quello appena descritto. È chiaro che i registri di un moderno microprocessore sono molto più specializzati e veloci di quelli di una comune calcolatrice. Come per la calcolatrice il più importante resta l'Accumulatore, che d'ora in poi chiameremo amichevolmente A. I registri X ed Y sono invece molto importanti durante le operazioni di trasferimento dei dati; infatti abbiamo detto che non è possibile scrivere o leggere direttamente il contenuto di una cella di memoria ma, come per le normali calcolatrici, è necessario passare attraverso l'accumulatore. Per scrivere quindi un numero nella cella $300 dovremo prima caricarlo in A e poi trasferirlo in $300. La lettura di una cella avviene trasferendone in A il contenuto. Se i dati da trasferire sono più di uno è possibile indicizzare la locazione di destinazione tramite i registri X ed Y che vengono perciò chiamati anche registri indice. Vediamo subito cosa vuol dire indicizzare un indirizzo di memoria. Diciamo che, per esempio, dobbiamo riempire di zeri le locazioni che vanno da $400 a $500 (nota: la RAM da $400 a $800 costituisce la mappa video, quindi tutto quello che viene scritto in questa zona compare automaticamente sullo schermo). Una prima soluzione potrebbe essere quella di caricare in A il numero $0 e poi di trasferire A in $400, poi in $401, poi in $402 e così via per 255 volte. L'istruzione che carica un numero in accumulatore è la LDA # «numero» (LoaD Accumulator, il # significa immediato ossia che quello che segue è proprio il dato), il cui codice Hex è $A9 (l'abbiamo già visto, ricordate?), mentre l'istruzione che trasferisce il contenuto di A in una cella di memoria è STA «indirizzo di due byte» (Storage Accumulator) codice Hex 8D. Quindi il nostro programma di trasferimento comincerebbe con:

```
*300 A9 00     LDA #$00
*302 8D 00 04  STA  $400
*305 8D 01 04  STA  $401
*307 8D 02 04  STA  $402
ecc.
```

Notiamo subito due cose: la prima è che gli indirizzi a due byte che seguono l'istruzione STA sono scritti, in memoria, in or-

**Figura 1**

| Routine | Indirizzo |
|---|---|
| ESC (F) | FC42 |
| ESC (E) | FC95 |
| HOME | FC58 |
| LINE FEED | FC66 |
| SCROLL | FC70 |
| CLRTOP (GR) | F836 |
| GRCLR | F832 |
| CLRHGR | F3F2 |
| COLORA HGR | F3F6 |
| STAMPA (Acc.) | FDED |
| INVERSE | FE80 |
| NORMAL | FE84 |
| A CAPO | FD8E |
| STAMPA BYTE (A) | FDDA |
| STAMPA NIBBLE (low A) | FDE3 |
| STAMPA 2 BYTE (A,X) | F941 |
| STAMPA 3 SPAZI | F948 |
| STAMPA X SPAZI | F94A |
| BELL | FBDD |
| GET (A) | FD1B |
| COLOR + 3 | F85F |
| PLOT (Y,A) | F800 |
| (A) = SCRN(Y,A) | F871 |
| (Y) = PDL(X) | FB1E |
| SALVA A,X,Y,P,S | FF4A |
| RIPRENDE REGISTRI | FF3F |
| STAMPA REG. ctrl(E) | FAD7 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| 2 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 3 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |
| 4 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 |
| 5 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 |
| 6 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 |
| 7 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | -126 | -125 | -124 | -123 | -122 | -121 | -120 | -119 | -118 | -117 | -116 | -115 | -114 | -113 | -112 | -111 |
| 9 | -110 | -109 | -108 | -107 | -106 | -105 | -104 | -103 | -102 | -101 | -100 | -99 | -98 | -97 | -96 | -95 |
| A | -94 | -93 | -92 | -91 | -90 | -89 | -88 | -87 | -86 | -85 | -84 | -83 | -82 | -81 | -80 | -79 |
| B | -78 | -77 | -76 | -75 | -74 | -73 | -72 | -71 | -70 | -69 | -68 | -67 | -66 | -65 | -64 | -63 |
| C | -62 | -61 | -60 | -59 | -58 | -57 | -56 | -55 | -54 | -53 | -52 | -51 | -50 | -49 | -48 | -47 |
| D | -46 | -45 | -44 | -43 | -42 | -41 | -40 | -39 | -38 | -37 | -36 | -35 | -34 | -33 | -32 | -31 |
| E | -30 | -29 | -28 | -27 | -26 | -25 | -24 | -23 | -22 | -21 | -20 | -19 | -18 | -17 | -16 | -15 |
| F | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 | 0 | 1 |

*Tabella 2 - Tabella dei salti relativi rispetto all'indirizzo del Branch.*

dine inverso, non 04 01 ma 01 04! Questa è una convenzione che vale per tutti gli indirizzi a due byte per cui occorre fare molta attenzione quando si scrive un programma in linguaggio macchina, per abitudine si usa la stessa convenzione anche quando si scrivono delle tabelle di indirizzi interne ad un programma (che gestendo direttamente noi potremo scrivere come più ci aggrada). Se si usa un assemblatore gli indirizzi possono essere scritti normalmente o possono essere definiti prima in una apposita tabella dove gli si possono assegnare dei nomi esattamente come per le variabili del Basic o le etichette delle programmabili.

La seconda cosa da notare è che i numeri a sinistra, che corrispondono alle locazioni di memoria in cui risiede il programma, non sono successivi, come i numeri di riga nel Basic, ma dipendono dalla lunghezza dell'istruzione. Questi numeri prendono il nome di Program Counter ed è a questi numeri che bisogna fare riferimento nelle istruzioni di salto.

Torniamo al nostro programma; appare subito evidente che il metodo scelto non è certamente quello più comodo né quello che occupa meno memoria. La soluzione sarebbe di usare una sorta di ciclo FOR NEXT proprio come faremmo in Basic. In alcuni elaboratori esiste una istruzione del genere, ma non nel 6502. Esiste però il registro X (che non abbiamo usato) e una istruzione simile alla IF che ci consente di uscire da un loop. Vediamo come possiamo usarle. Il registro X può servire, come abbiamo anticipato, ad indicizzare un indirizzo di memoria; questo significa che, cambiando l'istruzione STA «assoluta» con una STA «locazione assoluta», X è possibile dire al microprocessore che l'indirizzo in cui vogliamo scrivere non è quello che segue il codice STA ma lo stesso più il contenuto del registro X. Nella tabella 1, che consigliamo di fotocopiare e custodire gelosamente, troviamo che alla istruzione STA, — indirizzamento assoluto, X — corrisponde il codice $9D. Ci servirà anche una istruzione per mettere in X il valore 255 (quante volte dobbiamo eseguire il ciclo) e una per decrementare X, sempre dalla tabella troviamo per LDX (LoaD X) immediato (#) il codice $A2 e per DEX (Decrementa X) il codice $CA.

Per l'uscita dal ciclo ci serve un nuovo tipo di istruzione: il salto condizionato. Ovvero: se X è diverso da Zero allora vai a...

Di questi salti ne esistono vari tipi che vedremo via via che ci serviranno, tutti i salti condizionati lavorano su uno speciale registro del 6502 (uno di quelli di cui abbiamo accennato prima), il Processor Status, d'ora in poi P. Lo status è un registro a otto bit ma ciascuno di questi è visto dal microprocessore come un flag del fatto che si sia verificato un certo evento o che debba ricordarsi una certa predisposizione. Il registro P contiene, in ordine decrescente dal bit 7 al bit 0, i seguenti flag:

| bit | flag | significato |
|---|---|---|
| 7 | N | dato negativo (bit 7 del dato = 1) |
| 6 | V | overflow (riporto tra il bit 6 e il bit 7) |
| 5 | | non usato |
| 4 | B | comando di Break |
| 3 | D | modo di calcolo decimale |
| 2 | I | disabilita le interruzioni |
| 1 | Z | uguale a uno se il dato è Zero |
| 0 | C | è il Carry cioè il riporto o il prestito dell'Accumulatore |

A ciascuno di questi flag corrispondono varie istruzioni di salto condizionato. Nel nostro caso il flag che dovremo testare è il numero uno. Sul flag Z lavorano due salti (in inglese Branch: = diramazione) che permettono l'uscita in caso di Z = 1 o Z = 0. Se Z = 1 l'ultima operazione eseguita dal microprocessore ha dato un risultato uguale a zero, BEQ (Branch Equal) codice $F0, effettua il salto in caso di risultato uguale a zero; se Z = 0, e quindi il risultato è stato diverso da zero, il salto si ottiene con l'istruzione BNE (Branch Not Equal) codice $DO. Tutti i salti usano un particolare tipo di indirizzamento detto RELATIVO. Questo infatti dipende dal punto in cui l'istruzione di salto si trova. Dobbiamo quindi comunicare al Microprocessore non l'indirizzo assoluto in cui vogliamo andare, ma di quante istruzioni in avanti o indietro vogliamo spostarci. Questo porta un vantaggio e uno svantaggio; il vantaggio è che basta un solo byte per indicare una locazione invece di due e che anche spostando il programma in una diversa zona della memoria i salti risultano automaticamente corretti, lo svantaggio è costituito dal fatto che non è possibile raggiungere con un Branch locazioni di memoria che siano più lontane di 127 posti in avanti o indietro. In realtà lo spostamento effettivo varia tra +129 e -126 in quanto viene calcolato dalla prima locazione successiva all'istruzione di salto. Un ulteriore svantaggio lo incontra chi non possedendo un assemblatore che accetti un indirizzo per i salti e poi calcoli da solo la distanza relativa, deve fare i conti a mente o usare le apposite tabelle (vedi tabella 2).

Riprendendo l'esempio, il nostro programma dovrà 1) caricare il dato $0 in A; 2) caricare $FF (255) in X; 3) scrivere in $400 + X il contenuto di A; 4) decrementare X quindi: se X, = 0 allora finire, altrimenti ricominciare dal punto 3.

Ultima istruzione, l'uscita dal programma, è l'RTS che già abbiamo visto e il cui codice è $60.

Scriviamo il nuovo programma:

```
*300:A9 00 A2 FF 9D 00 04
*:CA DO FA 60
```

e listiamolo:

```
*300L
0300-  A9 00     LDA   #$00
0302-  A2 FF     LDX   #$FF
0304-  9D 00 04  STA   $0400,X
0307-  CA        DEX
0308-  DO FA     BNE   $0304
030A-  60        RTS
```

Facile no?

Avrete notato che il registro indice X è stato decrementato anziché incrementato come si sarebbe fatto in Basic, questo è dovuto al fatto che è molto più semplice, in linguaggio macchina, controllare il passaggio per lo zero, cui corrisponde un apposito flag, che confrontare due numeri e decidere se uno dei due è diventato maggiore dell'altro. Da notare che il programma siffatto presenta ancora un BUG (sono le pulci inglesi); infatti la locazione $400 non viene azzerata! Sareste in grado di correggerlo? (avete già tutti i mezzi). Già che ci siete sareste in grado, sapendo che il codice video dello spazio è $AO, di pulire tutto lo schermo che inizia a $400 e finisce a $800?

Arrivederci sul prossimo numero. MC