# IMPARIAMO A PROGRAMMARE IN assembler

di Valter Di Dio

## Quarta parte

*In questa quarta puntata concluderemo l'esplorazione dei tipi di indirizzamento che sono possibili con il microprocessore 6502. In caso abbiate dei dubbi mettetevi davanti al computer e fate delle prove; se ancora vi fosse sfuggito qualcosa non fatevi problemi a scrivere in redazione.*

*Oltre agli ultimi tipi di indirizzamento oggi faremo la conoscenza col Miniassembler che, presente nella prima serie di Apple, è purtroppo scomparso nelle ultime; niente paura, andremo a scovare dove si nasconde e lo riporteremo al lavoro.*

Quello che più contraddistingue il microprocessore 6502 dai suoi consimili è la incredibile varietà dei tipi di indirizzamento che questo consente. A prima vista sembrerebbe più utile avere a disposizione una serie di registri a sedici bit eventualmente separabili in coppie di registri a otto bit. I progettisti del 6502 hanno piuttosto preferito usare solo due registri a otto bit, aumentando invece la capacità di indirizzamento. Se questo consente una maggiore velocità di esecuzione in tutti quei casi in cui sono sufficienti i normali registri a otto bit, richiede peraltro una piccola fatica iniziale per il programmatore che, per poter lavorare su ampi spazi di memoria, deve imparare due strani tipi di indirizzamento dal nome alquanto sibillino: l'indirizzamento indicizzato indiretto e quello indiretto indicizzato.

Questi due tipi di indirizzamento suppliscono alla impossibilità di indirizzare indicizzatamente più di 256 locazioni di memoria contigue o di accedere sempre con un registro indice a dati che distino tra loro più dei famigerati 256 byte. Il loro uso, una volta capito bene il meccanismo con cui lavorano, consente di superare il confine dei 256 byte e di spaziare per tutta la memoria senza usare programmi che si automodificano.

Facciamo prima un breve riepilogo dei tipi di indirizzamento usati finora e del loro significato.

Indirizzamento implicito: significa che in pratica non si indirizza nulla dal momento che l'istruzione è fine a se stessa oppure specifica automaticamente il registro su cui lavora; ad esempio INX (incrementa X) o CLC (clear Carry).

Indirizzamento immediato: il dato si trova nella locazione di memoria immediatamente successiva al codice istruzione; ad esempio LDA #A0 equivale in memoria ad A9 A0, dove A9 è il codice istruzione ed A0 il dato.

Indirizzamento assoluto: i due valori che seguono il codice istruzione indicano rispettivamente la parte bassa e la parte alta dell'indirizzo di memoria che contiene il dato; per cui 8D 00 03 significa STA (Storage Accumulator) immagazzina il contenuto dell'Accumulatore nella locazione $0300.

Figura 1

```
0300-  A2 20     LDX   #$20
0302-  A9 D0     LDA   #$D0
0304-  85 81     STA   $81
0306-  85 82     STA   $82
0308-  A0 00     LDY   #$00
030A-  B1 81     LDA   ($81),Y
030C-  10 03     BPL   $0311
030E-  CA        DEX
030F-  F0 09     BEQ   $031A
0311-  E6 81     INC   $81
0313-  D0 02     BNE   $0317
0315-  E6 82     INC   $82
0317-  4C 0A 03  JMP   $030A
031A-  C8        INY
031B-  B1 81     LDA   ($81),Y
031D-  48        PHA
031E-  20 5C DB  JSR   $DB5C
0321-  68        PLA
0322-  10 F6     BPL   $031A
0324-  60        RTS
```

Indirizzamento Zero Page: è del tutto simile al precedente salvo che la parte alta dell'indirizzo di memoria viene omessa ed è assunta automaticamente uguale a zero; quindi A6 45 vuol dire carica nel registro X (LDX) il contenuto della locazione $0045.

Indirizzamento assoluto, X o Y: in questo caso al valore che segue il codice istruzione deve essere sommato il contenuto attuale del registro indice X, o Y a seconda del caso; il risultato indica la locazione definitiva di memoria in cui andare a prelevare il dato; per esempio FE A0 04 vuol dire incrementa il contenuto della locazione che si ottiene sommando a $04A0 il valore attuale di X, mettiamo che sia $10, per cui la locazione effettivamente incrementata diventa la $04B0.

Indirizzamento Zero Page, X (Y): è naturalmente simile a quello assoluto salvo che al solito la parte alta dell'indirizzo base vale zero ed è sottintesa. Da notare che solo il registro X può indicizzare la pagina zero, tranne nel caso in cui si debba indicizzare una operazione che coinvolge il registro X; nel qual caso si può usare il registro Y come indice. Non esiste quindi una LDA $F1,Y ma esiste la LDX $F1,Y.

Indirizzamento relativo: si usa nelle istruzioni di diramazione (branch) e significa che il valore che segue il codice istruzione viene interpretato come un intero con segno e sommato al valore attuale del contatore di programma, consentendo spostamenti in avanti o indietro relativi alla posizione della istruzione di diramazione. Per cui BEQ $04 significa salta avanti di 4 byte se l'ultimo risultato era zero.

Tutti questi tipi di indirizzamento li abbiamo già usati, magari senza accorgercene, negli esempi delle puntate precedenti. Se qualcuno nel frattempo si è andato a guardare la tabella 1 di pagina 57 sul numero 21, avrà scoperto che nella testata "indirizzamenti" restano ancora tre tipi sconosciuti: appunto l'indicizzato indiretto, l'indiretto indicizzato e l'indiretto. Quest'ultimo però è relativo solo alla istruzione di salto incondizionato JMP (GOTO) e significa semplicemente che nei due byte che seguono il codice operativo (6C) si trova l'indirizzo del primo di due byte che contengono la vera destinazione finale del salto (più complicato a dirsi che a farsi). Infatti 6C 00 03 che viene disassemblato in JMP $(0300), non salta a $300 ma a quello che trova scritto in $300 e $301.

### Rimbocchiamoci le maniche

Siamo finalmente arrivati ai due più potenti tipi di indirizzamento, gli indiretti indicizzati. Iniziamo con un problema pratico: vogliamo scrivere sullo schermo la parola riservata del Basic che corrisponde ad una certa posizione della tabella Applesoft.

La tabella delle parole Basic si trova già in Rom a partire dalla locazione DØDØ ma supera abbondantemente i 256 byte. Le parole riservate del Basic (tipo PRINT, END, INPUT ecc.) sono scrittte in ASCII tranne l'ultimo carattere che ha il bit di segno settato (un ASCII negativo ?!). Per accedere alla tabella potremmo usare una LDA $DØDØ,X ma questo non ci consentirebbe di esplorare le locazioni che si trovano oltre D1CF per il semplice fatto che X non può superare $FF. Si potrebbe ovviare al problema controllando X e, nel caso di superamento del confine, saltare ad un'altra routine che partendo da D1D0 acceda ad altre 256 locazioni e così via.

Risulta evidente che per leggere una tabella di soli 2K di memoria sarebbero necessarie 8 routine simili in cascata e addirittura ne servirebbero 32 solo per pulire una delle pagine grafiche in alta risoluzione.

A questo punto dobbiamo per forza ricorrere all'indirizzamento indiretto. In generale per indirizzamento indiretto si intende il fatto che quanto segue il codice istruzione non è il vero indirizzo finale ma l'indirizzo di una coppia di locazioni che contiene l'indirizzo finale. In questo modo anche se il programma risiede in Rom è possibile mandarlo a leggere un indirizzo

in Ram dove può essere facilmente manipolato. Per aumentare la velocità di accesso e per risparmiare un byte nell'istruzione si usa, per questo tipo di indirizzamento, la pagina zero. Oltre a essere indiretto questo indirizzo può anche essere indicizzato e cioè modificato dal contenuto di uno dei due registri indice X o Y. La modifica di questo indirizzo può essere fatta in due momenti, ricordiamo infatti che il microprocessore effettua due accessi in memoria, il primo quando va a leggere in pagina zero la locazione definitiva, il secondo quando appunto accede a quest'ultima. Nel 6502 è possibile indicizzare uno o l'altro di questi due accessi (non tutti e due) a seconda del registro indice utilizzato: X indicizza l'accesso alla pagina zero, Y quello alla locazione finale.

Per cui un indirizzamento di tipo LDA $(10,X) consente di accedere ad una tabella di salti che si trova in pagina zero a partire dalle locazioni $10 e $11 e proseguendo con $12 e $13, $14 e $15 e così via, determinando tramite il registro X quale coppia di locazioni contiene il puntatore al dato da caricare in Accumulatore (badate che il registro X deve essere incrementato di due per ogni coppia).

Un indirizzamento del tipo LDA $(81), Y accede ad una tabella il cui primo dato è puntato dal contenuto delle locazioni $81 e $82 più il valore attuale del registro Y. Come vedete oltre al modo in cui sono disposte le parentesi nel disassemblato ciò che principalmente distingue le due istruzioni è l'uso del registro X o Y. Il primo caso prende il nome di pre-indicizzazione o indicizzato (prima) indiretto (poi), il secondo si dice invece post-indicizzato o indiretto indicizzato.

È proprio l'indirizzamento indiretto indicizzato che ci consente di risolvere elegantemente il problema dell'accesso a tabelle di dati più lunghe di 256 byte.

Usando l'indirizzamento indiretto indicizzato è possibile accedere ai dati contenuti in una tabella qualsiasi in due differenti modi, a seconda che ci interessi la scansione completa, un byte dopo l'altro dal primo fino all'ultimo, oppure il prelievo di un certo numero di byte a partire da una locazione qualsiasi. Il primo metodo si usa di solito per stampare messaggi durante un programma in linguaggio macchina, il secondo serve invece per ritrovare delle stringhe all'interno di una tabella.

La differenza principale consiste nel fatto che per esplorare tutta una tabella, che in genere sarà più lunga di 256 byte, non si fa uso della post-indicizzazione e, mantenendo il registro Y azzerato, si incrementa, mediante un apposito sottoprogramma, la coppia di puntatori. Mentre nel caso della lettura di una parte della tabella (purché sia inferiore ai 256 byte) una volta trovato il punto di inizio incrementando via via la coppia di puntatori in pagina zero, si procede al prelievo dell'informazione incrementando il solo registro Y. Nel caso si abbia necessità di una scansione completa e si voglia però sfruttare la velocità di incremento del registro Y, si può usare una miscela di due metodi; ovvero si carica nei puntatori la locazione di start e si esplorano i primi 256 byte incrementando il registro Y; quando questo torna a zero si incrementa di uno la parte alta della coppia di puntatori, cioè il secondo dei due byte che si trovano in pagina zero. In questo modo si fa puntare l'indirizzamento indicizzato alla successiva pagina di memoria, dimodoché il successivo loop del registro Y esplora altri 256 byte. Occorrerà naturalmente un controllo di raggiungimento della fine della tabella che si potrà ricavare o direttamente dal riconoscimento di un byte di stop oppure dal superamento di una data locazione di memoria.

Il programma di figura 1, che deriva da una subroutine del Superlist, effettua la stampa di una parola riservata del Basic leggendola dalla tabella che si trova in ROM.

Vediamo come funziona: il registro X contiene il numero di comandi Basic che dobbiamo saltare prima di incontrare quello cercato. Quindi il primo comando carica in X il numero del comando; per esempio 5. Dopodiché dobbiamo prepararci ad usare l'indirizzamento indiretto indicizzato (LDA (hex), Y). Ci prepariamo allora due locazioni in pagina zero che dovranno contenere l'indirizzo di partenza della tabella. Per conformità al programma Superlist usiamo le locazioni $81 e $82. Dal momento che la tabella inizia a $DØDØ tutte e due le locazioni dovranno contenere $DØ; ricordate però che il $DØ della locazione $81 è effettivamente $DØ mentre quello della locazione $82 indica il numero della pagina di memoria (detto di solito parte alta dell'indirizzo in quanto è il byte di valore più elevato) e il suo valore deve essere moltiplicato per $FF. Una volta messe a posto, con la solita sequenza LDA... STA, le locazioni $81 e $82 passiamo ad azzerare il registro Y affinché punti proprio a $DØDØ. Ci serve ora il programma che incrementa di uno il puntatore ($81, $82). Immaginiamo per un momento che le locazioni $81 e $82 non siano in ordine inverso ma come di solito siamo abituati a disporre i numeri; in questo caso vediamo subito che per poter incrementare correttamente il contenuto del puntatore dobbiamo incrementare la locazione di destra ($82) finché non arriva a $FF, a questo punto il successivo incremento deve riportare a zero $82 ed effettuare il riporto di uno nella locazione $81. Dal momento che le locazioni possono contare solo fino a $FF un successivo incremento le riporta necessariamente a zero, quindi tutto regolare per la $82 mentre non esiste alcun riporto automatico tra celle di memoria; per cui una volta notato che la $82 è passata per lo zero dobbiamo essere noi ad incrementare di uno la locazione $81.

Figura 2

```
10  FOR I = 37120 TO 37437:A =  PEEK
    (I)
20  IF A = 32 OR A = 189 OR A = 2
    21 OR A = 217 THEN 50
30  NEXT
40  END
50 I = I + 1:A =  PEEK (I)
60  IF A < 148 THEN 30
70  POKE I,A + 100
80  GOTO 30
```

Figura 3

```
90ED-   A9 91       LDA   #$91
90EF-   85 74       STA   $74
90F1-   8D FA 03    STA   $03FA
90F4-   A9 00       LDA   #$00
90F6-   85 73       STA   $73
90F8-   A9 92       LDA   #$92
90FA-   8D F9 03    STA   $03F9
90FD-   4C 92 91    JMP   $9192
```

Dato che conosciamo già l'istruzione INC, sappiamo controllare il passaggio per zero grazie alla BEQ, niente di più semplice quindi realizzare il nostro incrementatore a sedici bit. Attenzione, ricordate che le locazioni sono scambiate fra loro e quindi invertite $81 e $82 nel programma. Dovreste essere tutti in grado di scrivere da soli tale subroutine che consta di soli sette byte RTS compreso; chi non ci riuscisse la trova in figura 1 a partire dalla locazione $311.

Come già accennato le parole chiave del Basic sono scritte in memoria in ASCII e terminano con un carattere maggiore di $7F, per contare quindi le parole da saltare ci basta contare quanti byte negativi (se sono maggiori di $7F hanno il bit 7 uguale ad uno e questo significa presenza di segno negativo) abbiamo incontrato. Ecco allora finalmente la LDA ($81),Y che legge un byte, viene poi controllata la sua positività e nel caso affermativo si procede ad incrementare $81-$82 e si prende il carattere successivo, altrimenti vuol dire che si è incontrata la fine di una parola per cui si decrementa il registro X. Quando X è arrivato a zero abbiamo raggiunto la parola cercata e saltiamo alla routine $31A che la stampa. La routine di stampa sfrutta ora la post-indicizzazione per scorrere i byte della parola da stampare ed inviarli alla routine DB5C del Basic che corrisponde alla FDED del monitor e che si preoccuperà di scriverli sullo schermo o sulla stampante se è attivo il canale 1, o addirittura passandoli al DOS come comandi se i primi due caratteri sono un carriage return ($8D) seguito da un control D ($84).

Quindi lasciato fermo il puntatore in pagina zero, che punta al primo carattere da stampare, si incrementa di uno il registro Y (INY) fino ad incontrare un "carattere negativo" che ci indica la fine della stringa e, per ora, del programma. Prima del JSR DB5C troviamo una strana istruzione, la PHA che serve a spingere sullo stack il contenuto dell'accumulatore. Questo è necessario in quanto al ritorno della routine di stampa il contenuto dell'accumulatore risulta modificato e non è più possibile controllare se il carattere appena stampato è l'ultimo. Avremmo potuto mettere il contenuto di A in una qualsiasi

locazione ma si sarebbero impegnate più locazioni di memoria sia per il programma che per contenere A. Con l'uso dello stack si risolve agevolmente il problema in quanto basta una PHA per spingere l'accumulatore sullo stack e una PLA per andarlo a riprendere; due byte in tutto e un programma più pulito.

## Il Miniassembler

Nei primi Apple con l'Integer Basic residente in Rom era disponibile un assemblatore piuttosto rudimentale ma abbastanza comodo per scrivere in memoria dei programmi di una certa lunghezza senza dover continuamente controllare le tabelle dei codici esadecimali o quelle per il calcolo degli indirizzi relativi.

Il Miniassembler non consente comunque di assegnare etichette a particolari punti del programma né di dare dei nomi alle locazioni di memoria usate per depositarvi le nostre variabili. Un'ulteriore comodità dei veri assemblatori è la possibilità di inserire pezzi di programma in qualsiasi punto e di definire tutti gli indirizzi non in modo assoluto ma relativo alla locazione di origine del programma; consentendo così la possibilità di rilocare il programma in qualsiasi parte della memoria (naturalmente rifacendo l'assemblaggio), queste possibilità non sono purtroppo presenti nel Miniassembler.

Nonostante ciò l'uso del Miniassembler è abbastanza utile, almeno dal punto di vista didattico, per avere un'idea generale degli assemblatori. Per chi decidesse di gettare via i manuali del Basic e togliere le relative Rom dall'Apple per dedicarsi alla sola programmazione in linguaggio macchina non potrebbe assolutamente fare a meno di un assembler sofisticato (magari in Rom). Resta dunque il problema costituito dal fatto che nell'Applesoft, per motivi di spazio, è scomparso il Miniassembler. Chi ci ha seguito negli scorsi numeri avrà senz'altro letto l'articolo che ci consente di recuperare l'Integer Basic che si trova nel disco Master e di come poterlo usare in Ram insieme all'Applesoft. Vista però la scomodità di avere il Miniassembler mescolato all'Integer Basic e soprattutto per il fatto che così si perde buona parte della Ram (da $6500 in poi), abbiamo deciso di estrarre il Miniassembler da lì e farlo diventare un programma a sé, visto oltretutto che è lungo appena 160 byte.

## Come recuperare il Miniassembler

La prima cosa da fare è di effettuare il bootstrap con il dischetto dell'Integer Basic preparato secondo le istruzioni che si trovano a pagina 72 del numero 18 di MC. A questo punto se provate a fare un *9266G vi comparirà il prompt del Miniassembler costituito dal punto esclamativo (!). Questo vi garantisce che il Miniassembler è in memoria e funziona bene. Per uscire dal Miniassembler o battete semplicemente il Reset oppure sfruttate il fatto

*90ED.923D — Figura 4

```
90ED- A9 91 85
90F0- 74 8D FA 03 A9 00 85 73
90F8- A9 92 8D F9 03 4C 92 91
9100- E9 81 4A D0 14 A4 3F A6
9108- 3E D0 01 88 CA 8A 18 E5
9110- 3A 85 3E 10 01 C8 98 E5
9118- 3B D0 6B A4 2F B9 3D 00
9120- 91 3A 88 10 F8 20 1A FC
9128- 20 1A FC 20 D0 F8 20 53
9130- F9 84 3B 85 3A 4C 95 91
9138- 20 BE FF A4 34 20 A7 FF
9140- 84 34 A0 17 88 30 4B D9
9148- CC FF D0 F8 C0 15 D0 E8
9150- A5 31 A0 00 C6 34 20 00
9158- FE 4C 95 91 A5 3D 20 8E
9160- F8 AA BD 00 FA C5 42 D0
9168- 13 BD C0 F9 C5 43 D0 0C
9170- A5 44 A4 2E C0 9D F0 88
9178- C5 2E F0 9F C6 3D D0 DC
9180- E6 44 C6 35 F0 D6 A4 34
9188- 98 AA 20 4A F9 A9 DE 20
9190- ED FD 20 3A FF A9 A1 85
9198- 33 20 67 FD 20 C7 FF AD
91A0- 00 02 C9 A0 F0 13 C8 C9
91A8- A4 F0 92 88 20 A7 FF C9
91B0- 93 D0 D5 8A F0 D2 20 78
91B8- FE A9 03 85 3D 20 34 92
91C0- 0A E9 BE C9 C2 90 C1 0A
91C8- 0A A2 04 0A 26 42 26 43
91D0- CA 10 F8 C6 3D F0 F4 10
91D8- E4 A2 05 20 34 92 84 34
91E0- DD B4 F9 D0 13 20 34 92
91E8- DD BA F9 F0 0D BD BA F9
91F0- F0 07 C9 A4 F0 03 A4 34
91F8- 18 88 26 44 E0 03 D0 0D
9200- 20 A7 FF A5 3F F0 01 E8
9208- 86 35 A2 03 88 86 3D CA
9210- 10 C9 A5 44 0A 0A 05 35
9218- C9 20 B0 06 A6 35 F0 02
9220- 09 80 85 44 84 34 B9 00
9228- 02 C9 BB F0 04 C9 BD D0
9230- 80 4C 5C 91 B9 00 02 C8
9238- C9 A0 F0 F8 60 00
```

che dal Miniassembler è possibile eseguire qualsiasi comando del Monitor semplicemente facendolo precedere del segno del dollaro ($); quindi per uscire dal Miniassembler potete battere !$FF69G
e vi ritroverete nel Monitor oppure
!$<CTRL>C<RETURN>
per tornare al Basic corrente.

Una volta certi che il Miniassembler esiste e funziona possiamo estrarlo dall'Integer Basic. Questo si ottiene semplicemente mettendo nel drive un disco qualsiasi (avrete un disco che usate per le prove!) e battendo: BSAVE MINI,A$9100,L$140

La lunghezza è leggermente eccedente ma questo per ora non importa.

Rifate ora il Boot con un disco normale in modo da ripulire la memoria dall'Integer Basic e dal DOS modificato. Ricaricate il programma MINI (BLOAD MINI) e provate a disassemblarlo. Scoprirete così degli strani JSR a locazioni tipo $981A o $94D0: derivano dal fatto che l'Integer Basic per sua comodità si porta appresso anche una copia del Monitor. Occorre quindi modificare tutte le istruzioni che indirizzano le routine del Monitor. Queste si riconoscono dal fatto che gli indirizzi sono compresi tra $9400 e $9BFF. Il programma in Basic di figura 2 effettua la conversione degli indirizzi sommando 100 (dec.) a quelli maggiori di $9400. Una ulteriore modifica consiste nell'aggiungere davanti al Miniassembler una routine che abbassa HIMEM a $9100 e mette nei puntatori del control Y che sono $3F9 e $3FA il punto di entrata del Miniassember ovvero $9192. La routine si trova in figura 3 e dovrebbe essere banale per tutti coloro che ci seguono. Comunque per maggiore chiarezza vi diamo alcune spiegazioni: le locazioni $73 e $74 contengono il valore della massima locazione di memoria accessibile al Basic, sostituendo il contenuto di queste locazioni con $00 e $91 proteggiamo il Miniassember da istruzioni accidentali da parte del Basic, poi con lo stesso principio spostiamo a $9192 il puntatore del JSR relativo al ctrl Y cosicché sia possibile passare al Miniassembler (dal Monitor) premendo <CTRL> Y.

Salvate il tutto su disco battendo:
BSAVE
MINIASSEMBLER,A$90ED,L$150
Per coloro che preferiscono copiarsi direttamente in memoria il Miniassembler diamo il dump di memoria relativo in fig. 4.

## Come si usa il Miniassembler

Una volta caricato il Miniassembler e lanciatolo in esecuzione con BRUN MINIASSEMBLER, vi trovate con il punto esclamativo come prompt. Per iniziare a scrivere un programma battete l'indirizzo di partenza seguito dai duepunti e dal mnemonico della prima istruzione. Per esempio: per mettere in $300 l'istruzione LDA #$90, scrivete: !300:LDA #90 <return>

Per proseguire l'inserimento battete lo spazio e poi direttamente la successiva istruzione. L'indirizzamento zero page viene riconosciuto dal fatto che l'indirizzo è ad un solo byte mentre tutte le altre istruzioni si scrivono nella stessa maniera del disassemblato del Monitor. Le istruzioni di salto condizionato, ad esempio BEQ, richiedono solo l'indirizzo assoluto cui saltare; il Miniassembler calcolerà automaticamente lo spostamento. Il sintax error è costituito da un proprio sotto al carattere errato; a volte però l'errore è altrove sulla stessa riga, considerate il solo come un avvertimento che qualcosa è andata male nell'interpretazione e ricontrollate tutta l'istruzione. Ogni volta che l'input ha avuto successo il Miniassembler presenterà subito il disassemblato completo dell'istruzione sicché si possa controllarne l'esattezza.

## Conclusioni

Nell'ultima puntata vi avevamo promesso una routine che permette di pulire lo schermo in alta risoluzione in 42 millesimi di secondo. Per motivi di spazio non è stato possibile inserirla in questa sede ed è finita nella rubrica del software Apple. MC

**Errata corrige:**
Programmare in Assembler, N. 21 pagina 57. Nella tabella 1 il codice della PHP è 8 invece di 6.