IMPARIAMO A PROGRAMMARE IN assembler

di Valter Di Dio

La puntata del mese scorso ha suscitato qualche perplessità; speriamo di eliminare ogni dubbio questo mese. In ogni caso abbiamo finalmente esaurito il discorso degli indirizzamenti che sono uno strumento indispensabile per chi voglia programmare in linguaggio macchina. Cominceremo ora il discorso sulle istruzioni vere e proprie: cosa fanno, come lo fanno e in quanto tempo. Utilizzeremo d'ora in poi solo i codici mnemonici disinteressandoci del corrispondente valore esadecimale e programmeremo in pratica come se utilizzassimo un Assemblatore; con tanto di Label, indirizzi Letterali e nomi per le locazioni che contengono variabili.

Questo permetterà una più facile comprensione dei programmi di prova anche per chi non avesse un Apple o addirittura possedesse una macchina basata su un diverso microprocessore (ricordiamo però che in questo caso tutto il discorso sui registri interni e gli indirizzamenti fin qui fatto molto probabilmente non troverebbe riscontro nella struttura interna del microprocessore).

Ancora indirizzamenti indiretti

Gli indirizzamenti indicizzati indiretti e indiretti indicizzati hanno provocato alcune perplessità dovute soprattutto al fatto che è abbastanza difficile capire quando certi valori indicano una locazione o il contenuto di dette locazioni; altra complicazione nasce dal fatto che negli indirizzamenti indiretti viene tirata in ballo anche la locazione successiva, ma successiva a cosa? Anche questo ha ingenerato una certa confusione.

Per capirci meglio dobbiamo prima metterci d'accordo sui termini; innanzitutto gli indirizzi: questi possono essere a otto o sedici bit. Dal momento che i normali indirizzi vanno da 0 a $FFFF (65534) evidentemente sono a sedici bit e necessitano quindi di due byte per essere memorizzati: un byte conterrà la parte alta dell'indirizzo, cioè quella relativa alla pagina di memoria, l'altro conterrà la parte bassa ovvero la cella di memoria della suddetta pagina che contiene il dato. Se parliamo di indirizzi a otto bit questi non possono essere altro che in pagina Zero, dato che con otto bit (un byte) si arriva a contare solo fino a $FF (255).

Il dato che segue un'istruzione è in genere un indirizzo tranne nel caso che sia preceduto dal # (simbolo inglese della parola NUMBER), nel qual caso è ovviamente un numero!

Nelle istruzioni in cui il dato è contenuto tra parentesi (per esempio LDA (88),Y) questo andrà interpretato come: l'indirizzo contenuto nella locazione NN. Ma una locazione sola non può contenere un indirizzo che è a sedici bit, gli otto restanti trovano allora posto nella locazione successiva: la NN + 1.

La tabella 1 dovrebbe a questo punto dissipare ogni dubbio sul modo in cui il 6502 calcola gli indirizzi indiretti indicizzati e indicizzati indiretti.

Le routine di entrata uscita

Nei programmi fin qui utilizzati per gli esempi non è mai stato necessario fornire alle routine informazioni supplementari e non ci siamo quindi mai occupati del problema della tastiera. Come si può leggere da un programma in linguaggio macchina il tasto premuto dall'operatore e come facciamo a sapere se è stato premuto un tasto?

A questo punto la risposta dipende dal tipo di macchina usata ma il discosrso è facilmente generalizzabile. Come prima cosa occorre che qualcosa ci informi dell'avvenuta pressione di un tasto; vediamo come è stato risolto questo problema dai costruttori di computer.

Nell'Apple, la pressione di un tasto provoca la comparsa di un uno nel bit di segno della locazione C000; questo blocca la tastiera che non accetta più dati finché il detto bit, chiamato strobe, non venga azzerato cosa che si ottiene semplicemente indirizzando la locazione C010. I restanti sette bit della locazione C000 equivalgono inoltre al codice ASCII del tasto premuto.

Nel Commodore 64 esiste un buffer di tastiera, lungo fino a dieci byte, che viene automaticamente riempito dal microprocessore senza intervento da parte del programmatore (il discorso è in realtà un pochino più complesso e coinvolge la gestione degli interrupt per cui evitiamo per ora di scendere nei particolari). Quando si preme un tasto il codice ASCII corrispondente viene depositato nel buffer di tastiera e viene aggiornato un puntatore all'ultimo carattere del buffer: se questo puntatore è maggiore di zero nel buffer c'è qualcosa, se è uguale a dieci è possibile che ci siamo persi parte del messaggio battuto da tastiera. Una volta letto il contenuto del buffer basta rimettere a zero il puntatore per riabilitarne la funzionalità.

Nelle tastiere esterne il principio è simile a quello usato dall'Apple solo che invece dello strobe si usa, di solito, mandare al microprocessore una richiesta di interrupt; il microprocessore, se non sono in corso

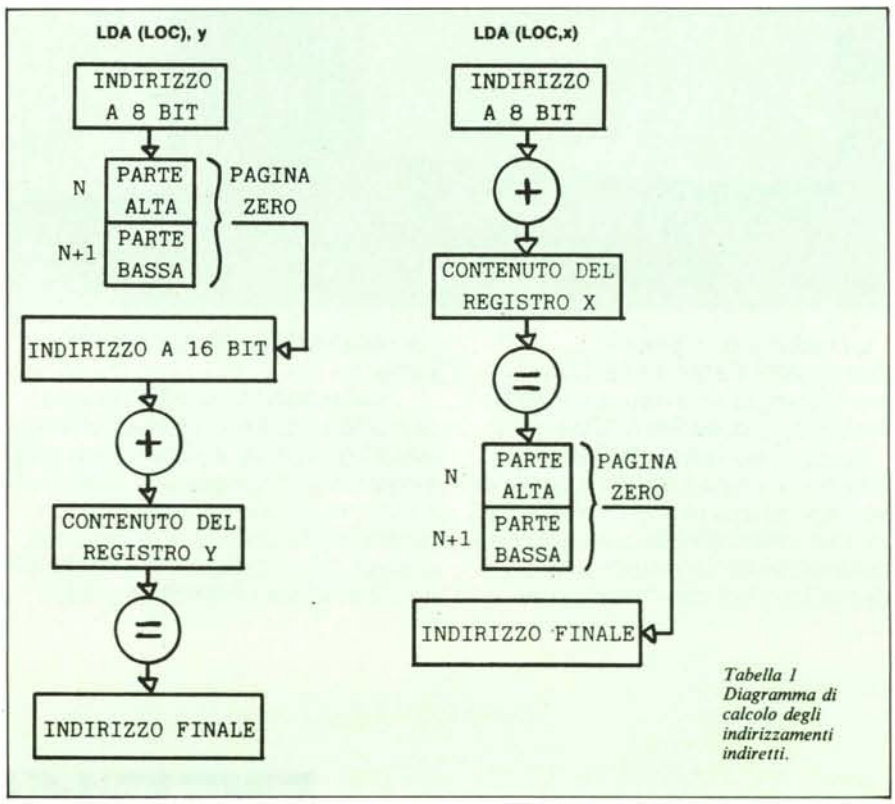

Tabella 1 Diagramma di calcolo degli indirizzamenti indiretti.

interruzioni a priorità più elevata (ad esempio da parte dei dischi), abbandona il suo lavoro e si dedica alla acquisizione del dato proveniente dall'esterno, dopodiché riprende il suo compito.

Nelle tastiere intelligenti un secondo microprocessore, a volte con una propria

```
0300-  A9 BE     LDA   #$BE
0302-  20 ED FD  JSR   $FDED
0305-  2C 00 C0  BIT   $C000
0308-  10 0A     BPL   $0314
030A-  AD 00 C0  LDA   $C000
030D-  2C 10 C0  BIT   $C010
0310-  20 ED FD  JSR   $FDED
0313-  60        RTS
0314-  20 30 03  JSR   $0330
0317-  A9 88     LDA   #$88
0319-  20 ED FD  JSR   $FDED
031C-  A9 A0     LDA   #$A0
031E-  20 ED FD  JSR   $FDED
0321-  A9 88     LDA   #$88
0323-  20 ED FD  JSR   $FDED
0326-  20 30 03  JSR   $0330
0329-  4C 00 03  JMP   $0300

0330-  A2 FF     LDX   #$FF
0332-  A0 FF     LDY   #$FF
0334-  88        DEY
0335-  D0 FD     BNE   $0334
0337-  CA        DEX
0338-  D0 F8     BNE   $0332
033A-  60        RTS
```

Figura 1 - Esempio di programma per la gestione di un cursore lampeggiante su Apple. La routine 330 corrisponde alla RITARDO dell'esempio in Assembler.

RAM, si occupa della gestione dei dispositivi di ingresso e di uscita e comunica poi col microprocessore principale sotto controllo di quest'ultimo.

Restando nel campo dei personal più comuni, Apple e Commodore, la lettura di un dato da tastiera si risolve in un controllo di Strobe e, se è il caso, nella lettura del contenuto di una o più celle di memoria.

Esempio:
lettura tastiera Apple

```
    LDA TASTO
    BPL no
    BIT CLRSTROBE
    no RTS
```

dove TASTO vale C000 e CLRSTROBE è la locazione C010.
lettura tastiera tipo Commodore 64

```
    LDX PUNTATORE
    BEQ no
    LDA BUFFER,X
    DEC PUNTATORE
    no RTS
```

dove PUNTATORE è la locazione che contiene il puntatore all'ultimo carattere del buffer e BUFFER è la prima locazione del buffer di tastiera. Da notare che questa routine preleva solo l'ultimo carattere presente nel buffer, occorrerà quindi richiamarla più volte se si vuole scaricare tutto il contenuto del buffer di tastiera.

Parliamo a questo punto della strana istruzione BIT che ci è servita a pulire lo strobe dell'Apple. La BIT serve per testare rapidamente alcuni bit di una locazione qualsiasi. Il test avviene in due fasi: viene eseguito l'AND tra la memoria e l'accumulatore, il risultato non è immagazzinato ma influenza il flag Z che vale 1 se il confronto è soddisfatto e 0 altrimenti. Poi vengono trasferiti i bit 7 e 6 della memoria rispettivamente nei flag N e V. Nel nostro caso la BIT ci è servita solo per indirizzare la locazione C010 senza influenzare il contenuto dei registri, ma avremmo anche potuto usarla per testare lo strobe senza modificare il contenuto dell'accumulatore:

```
    BIT TASTO
    BPL no
    LDA TASTO
    STA CLRSTROBE
    no RTS
```

in questo caso l'accumulatore non viene modificato se non è stato premuto alcun tasto, notate l'uso della STA per indirizzare la locazione C010 (CLRSTROBE).

Questi tipi di routine di lettura di un carattere non prevedono comunque alcun avviso per l'operatore: cursore lampeggiante o cose del genere, né l'arresto del programma e servono di solito per accettare dei comandi nei programmi di giochi o di editing di schermo. Qualora si desideri il cursore le precedenti routine vanno modificate in modo da girare dentro un loop che fa lampeggiare il cursore e da cui si esce solo a tasto premuto. Tra le routine del Monitor di tutti i tipi di macchina deve comunque esistere la routine di GETCHAR, simile alla GET del Basic, che serve a leggere un tasto premuto e si occupa contemporaneamente della gestione del cursore lampeggiante e, a volte, anche dell'eco video del carattere prelevato. Il codice ASCII del tasto premuto si trova di solito nell'Accumulatore al rientro dalla subroutine.

Sull'Apple, dove esiste un buffer di tastiera di 256 caratteri gestito da software, esiste anche la routine GETLINE che corrisponde praticamente alla INPUT del Basic; infatti usa la GETCHAR per riempire il Buffer di tastiera (tutta la pagina $2) e si ferma solo se incontra un RETURN o il buffer è pieno. Al ritorno il registro X contiene il numero di caratteri messi nel Buffer. La GETLINE (FD6A) permette anche la gestione del prompt il cui codice ASCII deve trovarsi nella locazione $33 e di tutti i normali comandi di Editing.

Rifare queste routine è piuttosto complicato e conviene solo in casi estremi; vediamo invece un semplice esempio di GET-

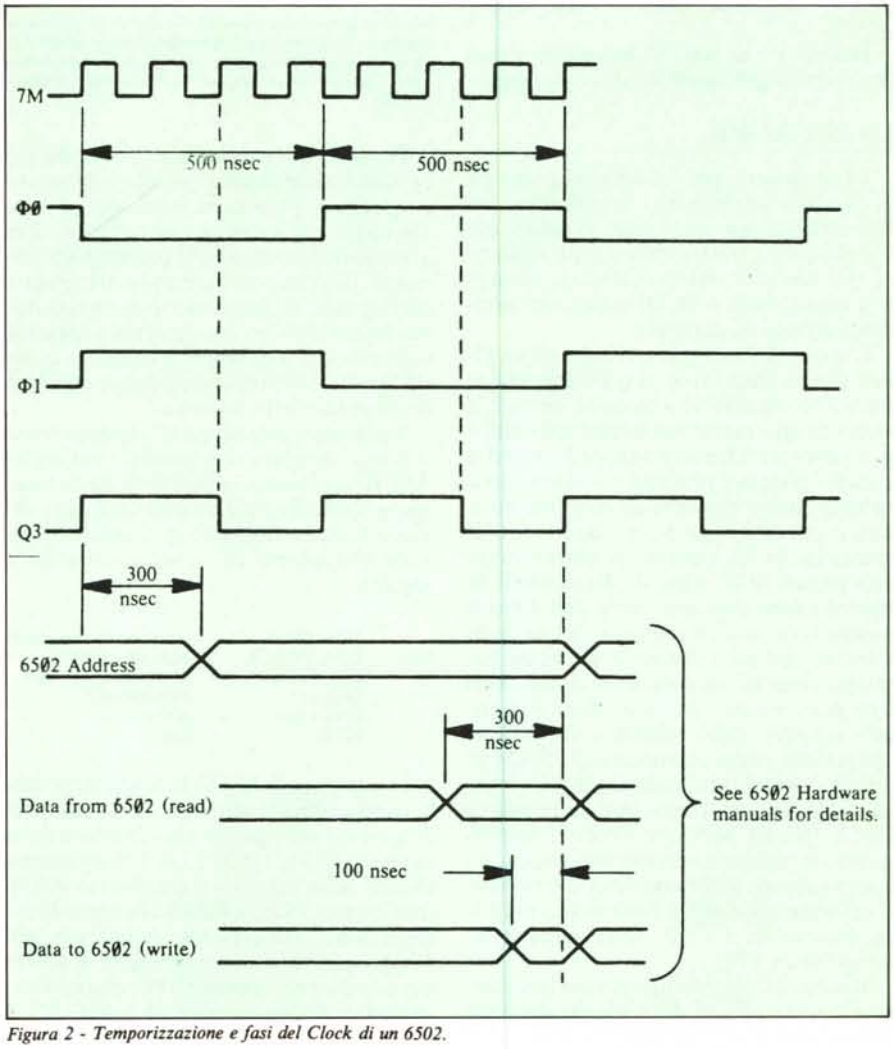

Figura 2 - Temporizzazione e fasi del Clock di un 6502.

CHAR con gestione software del cursore lampeggiante.

```
loop    LDA ">"          carica il cursore
        JSR PRINT        e lo stampa
        BIT TASTO        tasto premuto?
        BPL no
        BIT CLRSTRB      si, pulisci il bit 7
        LDA TASTO        leggi il codice
        JSR PRINT        stampalo sul video
        RTS              e ritorna
no      JSR RITARDO      aspetta un po'
        JSR BCKSPC       torna indietro
        LDA " "          carica lo spazio
        JSR PRINT        spegni il cursore
        JSR BCKSPC       ritorna indietro
        JSR RITARDO      aspetta ancora un po'
        JMP loop         e rifai da capo
```

Le routine PRINT, BaCKSPaCe e RITARDO dovranno naturalmente esistere da qualche parte, o nel Monitor o in un'altra parte del vostro programma. L'istruzione mnemonica LDA "carattere" che compare nel programma equivale ad una LDA #$xx, solo che sarà l'Assemblatore ad occuparsi di sostituire il carattere tra virgolette con il corrispondente codice ASCII. Purtroppo non tutti gli assembler in commercio dispongono di questa utilissima possibilità che risparmia al programmatore affannose ricerche sulle tabelle ASCII.

In figura 1 di pag. 73 trovate lo stesso programma già assemblato su un Apple.

L'uscita dei dati

Ci occupiamo per ora solo della stampa di caratteri alfanumerici tipo stringa; nel caso della stampa di risultati di calcoli occorrerà prima trasformare il dato numerico dal formato interno (binario, binario con segno, float o BCD) nella corrispondente stringa di caratteri!

L'uscita dei dati può avvenire sia su video che su stampante, il problema che si pone è molto diverso a seconda dei casi. Il video di una macchina basata sul 6502 è generalmente Memory Mapped ovvero a ciascun carattere presente sul video corrisponde biunivocamente un certo valore in una o più locazioni di una data zona di memoria. Se ad esempio vogliamo avere una pagina di 25 righe da 80 caratteri in bianco e nero dovremo avere 2000 byte di memoria destinati a contenere la mappa di schermo, nel caso volessimo anche dei caratteri colorati sarebbero necessari altri byte per contenere il colore relativo ai singoli caratteri dello schermo. Qualunque operazione venga compiuta sulla RAM di schermo viene automaticamente riportata sul video. Questa tecnica prende il nome di DMA (Direct Memory Access), accesso diretto in memoria, ovvero: una circuiteria esterna accede direttamente ai dati presenti nella memoria senza l'intervento del microprocessore, quindi senza perdita di tempo della CPU.

Il motivo di questo tipo di configurazione è dovuto ad una particolarità del 6502 che ne facilita notevolmente l'uso. Infatti il Clock di sistema viene diviso dal microprocessore in due fasi simmetriche $\phi 0$ e $\phi 1$ (vedi figura 2), il 6502 usa solo la fase 0 per gli indirizzi e i dati, mentre durante la fase 1 esegue delle operazioni interne che non necessitano dei BUS dati e indirizzi. È proprio durante la fase 1 che un'altra unità può accedere alla memoria senza disturbare il lavoro del microprocessore.

Anche per quanto riguarda l'uscita video conviene appoggiarsi sulle routine del Monitor a meno che non si voglia effettuare delle modifiche sostanziali.

```
0300-   A2 10        LDX   #$10
0302-   BD 50 03     LDA   $0350,X
0305-   9D 00 06     STA   $0600,X
0308-   CA           DEX
0309-   D0 F7        BNE   $0302
030B-   60           RTS


0350- A0 CD C3 AE CD C9 C3 D2
0358- CF C3 CF CD D0 D5 D4 C5
0360- D2
```

Figura 3 - Programma per la stampa diretta sul video di una stringa alfanumerica. La stringa è contenuta in memoria a partire dalla locazione 350 ed è in codice ASCII-VIDEO.

Per gestire correttamente il video occorre infatti controllare un numero abbastanza grande di parametri, basti solo pensare alle routine di scrolling e alla possibilità di avere varie finestre aperte contemporaneamente. Inoltre spesso la mappa di memoria corrisponde allo schermo in modo piuttosto disordinato per cui occorrono apposite subroutine al solo scopo di calcolare quale sia la cella di RAM che corrisponde ad un certo punto dello schermo.

Vediamo a solo scopo di esempio come si possa stampare una parola, i cui codici ASCII si trovano in memoria dalla locazione $350 alla $360, in una data zona del video di cui conosciamo già l'indirizzo iniziale e sappiamo che i successivi sono a seguire.

```
        LDX #LEN         carica lunghezza parola
loop    LDA TAB., X      legge un carattere
        STA RIGA, X      lo deposita sul video
        DEX              decrementa X
        BNE loop         rifai se >0
        RTS              fine.
```

In questo caso LEN è la lunghezzza della parola (deve essere minore del numero di caratteri per riga e in ogni caso non deve superare $FF), TABELLA è la locazione iniziale della tabella dei caratteri in RAM (nell'esempio $350), RIGA è la prima locazione di una qualsiasi riga di schermo (per l'Apple potrebbe essere la mappa di schermo o andare per tentativi). Per alcune macchine (leggi Commodore 64 o VIC 20) il programma così com'è non è sufficiente in quanto le mappe video sono due: una contiene i caratteri l'altra il colore. In questo caso il programma va modificato così:

```
        LDX              #$LEN
loop    LDA              Tabella. Caratteri, X
STA     RIGA. VIDEO, X
LDA                      Tabella. Colore, X
STA                      Riga. Colore, X
DEX
BNE                      loop
RTS
```

Abbastanza simile al precedente salvo che le tabelle devono essere 2; se si usa un solo colore si può sostituire la LDA COLORE, X con una LDA #COLORE.

In figura 3 trovate il primo programma pronto per girare su un Apple (tabella compresa!). Non dovrebbe essere difficile realizzare il secondo esempio una volta note le zone di RAM del video ($400-$800 per il 64) e del colore (55296 - 56320 del 64). Per il VIC 20 queste aree differiscono purtroppo a seconda della memoria disponibile. Come più volte detto occorre una buona conoscenza della propria macchina per poter programmare in Assembler ad un certo livello.

Per quanto riguarda invece la stampante, la gestione è simile a quella di una tastiera esterna con la differenza che il flusso dei dati è ora uscente. In pratica, per inviare un dato alla stampante dobbiamo testare una certa locazione per vedere se la stampante è pronta ad accettare il nostro dato; appena il test dà risultato positivo dovremo scrivere in un'altra locazione il valore che vogliamo inviare alla stampante. Questo va ripetuto per ciascun carattere.

Esempio:

```
printout LDA $00          legge il dato da inviare
loop     BIT READY        è pronta?
         BMI loop         no, aspetta.
         STA DATO         si, stampalo
         RTS              fatto, ritorna!
```

Dove READY è una locazione il cui bit di segno viene posto a 1 dall'interfaccia finché non riceve il segnale di ready dalla stampante, e DATO è la locazione da cui l'interfaccia leggerà il valore da inviare alla stampante. La locazione $00 all'inizio del programma è usata solo come appoggio e può essere una qualsiasi locazione. Per l'Apple con interfaccia parallela EPSON nello slot 1 le due locazioni sono: READY = $C1C1 e DATO = $C090.

Conclusioni

Ora che abbiamo esaurito il discorso degli indirizzamenti possiamo procedere in modo più spedito all'esplorazione delle istruzioni del 6502. Nella prossima puntata cominceremo una esposizione sistematica delle singole istruzioni, un po' come abbiamo fatto oggi per la BIT. Ci occuperemo dei primi semplici calcoli e vedremo in che modo ciascuna istruzione modifica il registro che contiene i flag.

Nel frattempo provate a buttar giù i programmini di questa puntata e provate magari a modificarli un po'. MC

IL TUO PRIMO COMPUTER

REBIT COMPUTER
A DIVISION OF GBC

ZX81

CON ALIMENTATORE

sinclair

Il computer più venduto nel mondo

£. 99.000

Il prezzo non è comprensivo di IVA