

# GIOCHI CON IL 6502

Tecniche di programmazione avanzate



Rodnay Zaks

EDIZIONE  
ITALIANA



GRUPPO  
EDITORIALE  
JACKSON





# **GIOCHI CON IL 6502**

**Tecniche di programmazione avanzate**

**di  
Rodnay  
Zaks**



**GRUPPO  
EDITORIALE  
JACKSON**

**Via Rosellini, 12  
20124 Milano**

**Tel. 680368 - 680054 - 68951/2/3/4/5**

## RINGRAZIAMENTI

L'autore desidera ringraziare Chris Williams ed Eric Novikoff per aver controllato tutti i programmi dei giochi e contribuito a numerosi miglioramenti.

L'autore è particolarmente grato ad Eric Novikoff per la sua assistenza in tutte le fasi di preparazione del manoscritto e per la meticolosa supervisione del testo finale.

### Nota

SYM è un marchio registrato dalla Synertek Systems, Inc.

KIM è un marchio registrato della MOS Technology, Inc.

AIM65 è un marchio registrato della Rockwell International, Inc.

"COMPUTERCHERT" e "GAMES BOARD" sono marchi registrati dalla Sybex Inc.

Disegno di copertina di Daniela Le Noury

Illustrazioni tecniche di Guy S. Orcutt e J. Trujillo Smith.

Si è cercato per quanto possibile, di fornire informazioni complete e rigorose. In ogni caso la Sybex non si assume alcuna responsabilità per il loro impiego; nemmeno al riguardo di infrazioni di brevetti e di altri diritti di terze parti che ne potrebbero derivare. I costruttori di apparecchiature non rilasciano alcuna autorizzazione su apparecchiature protette da brevetto o diritti di brevetto e si riservano la facoltà di cambiare, in qualunque momento la disposizione circuitale senza alcun preavviso.

In particolare sono soggetti a frequente cambiamento le caratteristiche tecniche e i prezzi. I confronti e le valutazioni sono presenti solo per il loro valore educativo ed i loro principi informativi. Per le specifiche esatte si rimanda il lettore ai dati del costruttore.

Copyright per l'edizione originale SYBEX Inc. 1980

Copyright per l'edizione originale SYBEX Inc. 1982

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca di Fiore, l'Ing. Sergio Zannoli e l'Ing. Roberto Pancaldi. Traduzione a cura di eds electronic data service - Bresso (MI).

Tutti i diritti sono riservati - Nessuna parte di questo libro può essere riprodotta posta in sistemi di archiviazione, trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopiatura etc. senza l'autorizzazione scritta dell'editore.

Stampato in Italia da  
S.p.A. Alberto Matarelli - Milano  
Stabilimento Grafico



## PREFAZIONE

*"Gli algoritmi complessi possono costituire  
un divertimento!"*

Spesso i programmatori considerano programmare alla stregua di un gioco, anche se non sono disposti ad ammetterlo. Al limite la programmazione di un computer può essere considerata un gioco intelligente.

Un programma è la proiezione della propria intelligenza ed abilità, e la scrittura di programmi di giochi costituisce un ulteriore divertimento aggiuntivo. Compunque la maggior parte dei giochi interessanti sono notevolmente complessi da realizzare e richiedono artifici di programmazione specifici.

Questo libro vi insegnerà come programmare una gamma completa di giochi, da quelli passivi (Musica) a quelli strategici (Tic-Tac-Toe). Nel processo di apprendimento su come programmare questi giochi affinerete artifici sulle tecniche d'ingresso/uscita, quali timer ed interrupt. Impiegherete inoltre diverse strutture dati e migliorerete o svilupperete i vostri artifici di programmazione a livello assembly.

Questo libro è stato concepito come *testo educativo*. Dopo la sua lettura dovreste essere in grado di creare programmi per altri giochi e di impiegare gli artifici di programmazione in altre applicazioni.

Se avete accesso ad una scheda microcomputer potete divertirvi sulla base dei risultati del vostro lavoro in tempi molto brevi. La lista dei programmi presentati in questo libro è relativa alla scheda SYM (della Synertek Systems), ma può essere adattata ad altri microcomputer basati sul 6502.

L'esecuzione di questi giochi richiederà una semplice ed economica "Scheda Giochi", che viene descritta al Capitolo 1. Per facilitare l'esecuzione del gioco è inoltre disponibile presso la Sybex nel formato SYM una "Cassetta Giochi".

I numerosi giochi studiati in questo libro comprendono: giochi musicali (MUSICA), giochi didattici (TRADUCI ed INDOVINA L'ESADECIMALE che vi insegneranno l'esadecimale), giochi che coinvolgono la logica (QUADRATI MAGICI), giochi di coordinazione (SPINNER), giochi di memoria (ECO), giochi di fortuna

(SLOT MACHINE), giochi di strategia (TIC-TAC-TOE) e giochi che coinvolgono diverse combinazioni di artifici (BLACKJACK).

Nella presentazione del programma di ciascun gioco viene seguito un formato comune che comprende:

1. Le regole del gioco.
2. Le istruzioni per eseguire un gioco tipico.
3. L'algoritmo o gli algoritmi (teoria del funzionamento).
4. Il programma: strutture dati, tecniche di programmazione, subroutine.

Nel corso del testo vengono inoltre suggerite variazioni ed esercizi.

In sostanza apprenderete dapprima come eseguire il gioco e successivamente come escogitare una possibile soluzione (l'algoritmo). Infine realizzerete l'implementazione reale di una versione programmata completa dell'algoritmo in linguaggio di livello assembly sul 6502, dedicando un'attenzione particolare alle strutture dati richiesti ed alle tecniche impiegate per una programmazione efficiente.

L'apprendimento della programmazione in linguaggio assembly è sempre stato tradizionalmente considerato come poco attraente e difficile. In realtà può essere divertente. Se avete familiarità con le tecniche elementari di programmazione, a livello del testo di riferimento - *Programmazione del 6502*, pure edito dal Gruppo Editoriale Jackson questo libro vi insegnerà le tecniche pratiche di programmazione in un contesto di gioco. Vengono integrati i concetti teorici analisi passo-passo dello sviluppo del programma. Le stesse tecniche e gli stessi concetti possono essere applicati a qualsiasi problema di programmazione, dal controllo industriale alle applicazioni commerciali.

Con l'augurio di un apprendimento della programmazione in modo altrettanto divertente quanto l'esecuzione dei giochi, vi chiedo di scrivervi se avete inventato, sviluppato e se conoscete altri giochi che potrebbero essere inclusi in un libro di giochi.

RODNAY ZAKS

# SOMMARIO

<b>PREFAZIONE</b> .....	<b>III</b>
<b>CAPITOLO 1 - INTRODUZIONE</b> .....	<b>1</b>
<i>La Scheda Giochi.</i>	
<b>CAPITOLO 2 - GIOCO MUSICALE</b> .....	<b>21</b>
<i>Esegue una sequenza massima di 255 note (13 note diverse e le registra automaticamente.</i>	
<b>CAPITOLO 3 - TRADUCI</b> .....	<b>43</b>
<i>Il computer visualizza un numero binario. A turno, ogni giocatore deve premere più velocemente possibile l'equivalente esadecimale. Colui che arriva prima a 10 vince. Previsto per due giocatori.</i>	
<b>CAPITOLO 4 - INDOVINA L'ESADECIMALE</b> .....	<b>63</b>
<i>Occorre indovinare un numero esadecimale di due digit generato dal computer. Il computer vi dirà la differenza tra questi numeri. Sono consentiti fino a 10 tentativi.</i>	
<b>CAPITOLO 5 - QUADRATO MAGICO</b> .....	<b>77</b>
<i>Accendete un quadrato perfetto sulla scheda. Ogni tasto inverte alcune strutture di LED. Sono richiesti artifici e logica.</i>	
<b>CAPITOLO 6 - SPINNER</b> .....	<b>91</b>
<i>Una luce ruota attorno ad un quadrato. Dovete catturarla premendo il tasto corrispondente. Ogni volta che riuscite, la rotazione diventerà più veloce. Un gioco di abilità.</i>	
<b>CAPITOLO 7 - SLOT MACHINE</b> .....	<b>105</b>
<i>Viene simulata una slot machine del tipo Las Vegas, con tre ruote rotanti. Provate la vostra fortuna.</i>	
<b>CAPITOLO 8 - ECO</b> .....	<b>145</b>
<i>Riconoscete duplicate una sequenza suono/luce (conosciuto anche come SIMON - un marchio registrato).</i>	
<b>CAPITOLO 9 - MINDBENDER</b> .....	<b>171</b>
<i>Giocate contro il "banco" (il computer) con un mazzo di 10 carte. Potete battere o fermarvi. Non andate in rovina!</i>	

<b>CAPITOLO 10 - BLACKJACK .....</b>	<b>201</b>
<i>Indovinate una sequenza di numeri generati dal computer. Esso vi dirà quanti digit sono corretti e nella posizione corretta (conosciuto anche come MASTERMIND - un marchio registrato).</i>	
<b>CAPITOLO 11 - TIC - TAC - TOE.....</b>	<b>230</b>
<i>Provate ad ottenere tre elementi in una riga prima del computer, in questo gioco di strategia. L'abilità del computer migliora con la vostra. Riuscite a batterlo?</i>	
<b>APPENDICE A - Istruzioni in ordine alfabetico del 6502 .....</b>	<b>301</b>
<b>APPENDICE B -</b>	
<b>Set di istruzioni del 6502: Esadecimale e Timing .....</b>	<b>302</b>

## CAPITOLO I

# INTRODUZIONE

### SCOPO

Questo libro è dedicato al programmatore che desidera apprendere le tecniche di programmazione avanzata con l'impiego del 6502. Naturalmente esso può essere impiegato da coloro che desiderano semplicemente eseguire giochi con la propria scheda basata sul 6502. L'impiego di questo libro per scopi didattici presuppone familiarità con il set di istruzioni del 6502 e delle tecniche di programmazione di base *Programmazione del 6502*. Si raccomanda inoltre una conoscenza di base delle tecniche d'ingresso/uscita. (*Applicazioni del 6502* del Gruppo Editoriale Jackson).

I giochi presentati nel corso del libro vanno da programmi semplici ad altri estremamente complessi. Per l'implementazione dei programmi, verranno proposti gli algoritmi e saranno progettate le strutture dati. Questo è il processo che segue qualsiasi programmatore rigoroso per la progettazione di una soluzione programmata ad un problema assegnato.

Normalmente i programmi di giochi non presentano alcun problema d'ingresso/uscita, a differenza di alcuni programmi di controllo industriale; comunque essi rappresentano spesso una vera sfida intellettuale per escogitare una strategia efficiente di soluzione. Inoltre tutti gli algoritmi ed i programmi presentati sono stati progettati per essere concisi in modo che possano risiedere in meno di 1 K di memoria.

Inoltre tutti i programmi sono stati correttamente realizzati e collaudati da diversi utenti che li hanno verificati esenti da errori. L'autore sarà grato per qualsiasi commento o suggerimento; che possa rivestire un qualche interesse per i lettori.

I programmi di questo libro possono essere impiegati per eseguire giochi reali. Essi richiedono l'impiego di una scheda basata sul 6502, come la scheda SYM (prodotta e brevettata dalla Synertek Systems) e richiedono l'implementazione di una semplice "Scheda Giochi". Questo capitolo fornisce una descrizione completa della Scheda Giochi mostrata in Fig. 1-1.

Questi programmi, così come sono presentati, possono essere eseguiti direttamente su una scheda SYM, ma possono essere facilmente adattati a qualsiasi altro computer basato sul 6502. Comunque, le linee d'ingresso/uscita disponibili sono normalmente specifiche del computer impiegato. I segmenti d'ingresso/uscita dei vari programmi devono essere quindi modificati conseguentemente. Naturalmente gli algoritmi e le tecniche di programmazione impiegati rimangono generalmente invariati.

Dopo la lettura di questo libro, specialmente se eseguite i programmi sulla Scheda Giochi, probabilmente sarete d'accordo nell'affermare:

"Gli algoritmi complessi possono essere divertenti!"

## **HARDWARE RICHIESTO**

Per eseguire i programmi presentati in questo libro su un microcomputer reale si può impiegare su SYM oppure un'altra scheda basata sul 6502. Inoltre è richiesta una Scheda Giochi addizionale. La Fig. 1-1 mostra una fotografia della Scheda Giochi. Si tratta, come dice il nome, di una scheda d'ingresso/uscita sulla quale possono essere eseguiti i giochi. La tastiera che appare sulla destra viene impiegata come ingresso alla scheda microcomputer, mentre i LED che appaiono sulla sinistra, vengono impiegati per visualizzare l'informazione inviata dal programma. Per ogni gioco proposto verrà spiegato l'impiego dei tasti e dei LED. Per ottenere effetti sonori, è stato connesso un altoparlante. Esso è stato montato in una cassa per migliorarne la qualità del suono. (Vedere Fig. 1-2).

La Scheda Giochi può essere facilmente realizzata in casa con un piccolo numero di componenti a basso costo, oppure si può richiedere alla Sybex. Poichè l'assemblaggio è abbastanza semplice, si consiglia vivamente al lettore interessato di eseguire personalmente l'implementazione della scheda, anche per acquisire la massima comprensione dell'hardware. D'altra parte, l'assemblaggio della Scheda Giochi non è indispensabile per l'impiego di questo libro. Essa consente semplicemente di estendere il livello di comprensione.

## **CONNESSIONE DEL SISTEMA**

Assumiamo che sia disponibile una scheda microcomputer basata sul 6502, come la scheda SYM ed una Scheda Giochi.

Questo paragrafo descrive come interconnettere gli elementi del sistema in modo che possano essere effettivamente realizzati i giochi descritti nei capitoli successivi. Se avete accesso a questo hardware tralasciate

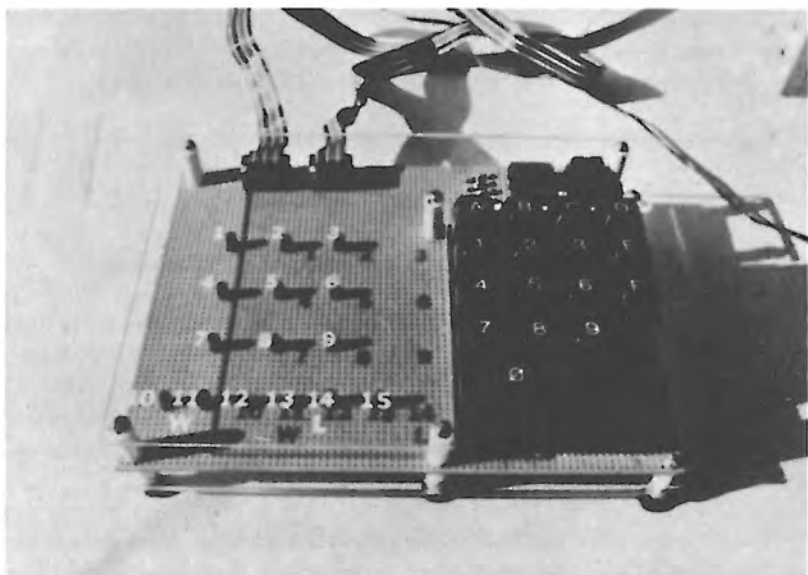


Figura 1.1: La scheda giochi



Figura 1.2: Un contenitore può essere impiegato per migliorare la qualità sonora



questo paragrafo. Comunque è possibile che in seguito dobbiate fare riferimento ad esso, per implementare i giochi descritti nel libro oppure per capire le tecniche d'interfacciamento o d'ingresso/uscita.

Sono richiesti quattro componenti essenziali:

- 1 - l'alimentatore
- 2 - la scheda SYM
- 3 - la Scheda Giochi
- 4 - un registratore a cassette (preferibilmente)

Il primo passo è di connettere l'alimentatore di sistema. Se non ne è già dotato, occorre connettere due gruppi di conduttori. (Vedere Fig. 1.3.) Il primo gruppo deve essere connesso ad un cavo di alimentazione. Il secondo gruppo, comprendente i conduttori di massa e 5V, deve essere connesso al connettore di alimentazione del SYM, sulla base delle specifiche del produttore.

Successivamente occorre procedere alla connessione fisica della Scheda Giochi al SYM: vengono impiegati i connettori A ed AA. (Vedere Fig. 1.4). Inoltre è presente il connettore per l'alimentazione.

Occorre fare attenzione ad inserire i connettori con l'orientamento corretto (normalmente il lato che porta la stampigliatura rivolto verso l'alto). In particolare un errore di connessione del connettore di alimentazione può causare sorprese molto molto spiacevoli, invece errori di connessione dei connettori di I/O sono normalmente meno catastrofici.

Infine, qualora sia utilizzato un registratore a cassette (vivamente consigliato), la scheda SYM deve essere connessa a tale registratore. Almeno dovrebbero essere connessi i terminali "monitor" e "cuffia" e, preferibilmente, il conduttore "remote". Se si vuole memorizzare su nastro i nuovi programmi, occorre connettere anche i terminali "registrazione" e "microfono". (Vedere Fig. 1.5). I dettagli per queste connessioni sono forniti sul manuale del SYM.

A questo punto il sistema è pronto per l'impiego. (Vedere Fig. 1.6). Se disponete di una delle cassette giochi (disponibili separatamente presso la Sybex) caricate semplicemente la cassetta nel registratore. Premete il tasto RST dopo aver alimentato il SYM e caricate il gioco richiesto nel SYM stesso. A questo punto si può iniziare il gioco.

Altrimenti occorre caricare il codice oggetto, in codice esadecimale, sulla tastiera del SYM, relativamente al gioco richiesto. Tutti i giochi iniziano saltando alla locazione 200 ("GO 200").

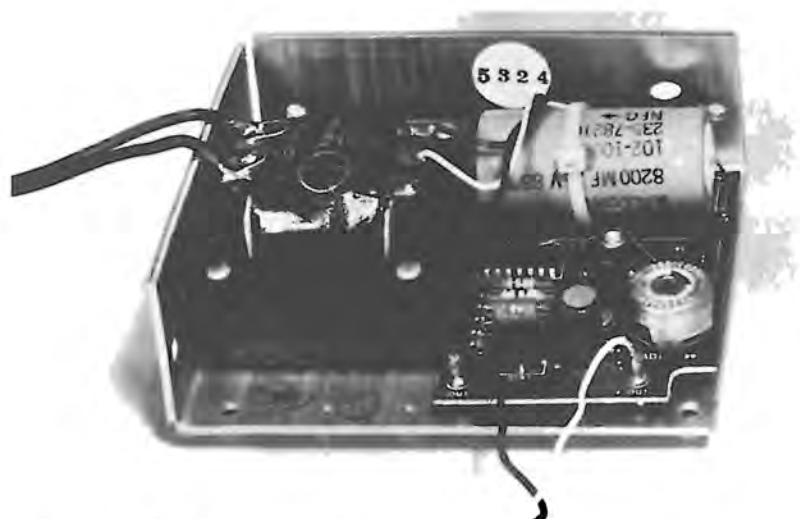


Figura 1.3: Due conduttori devono essere connessi all'alimentatore

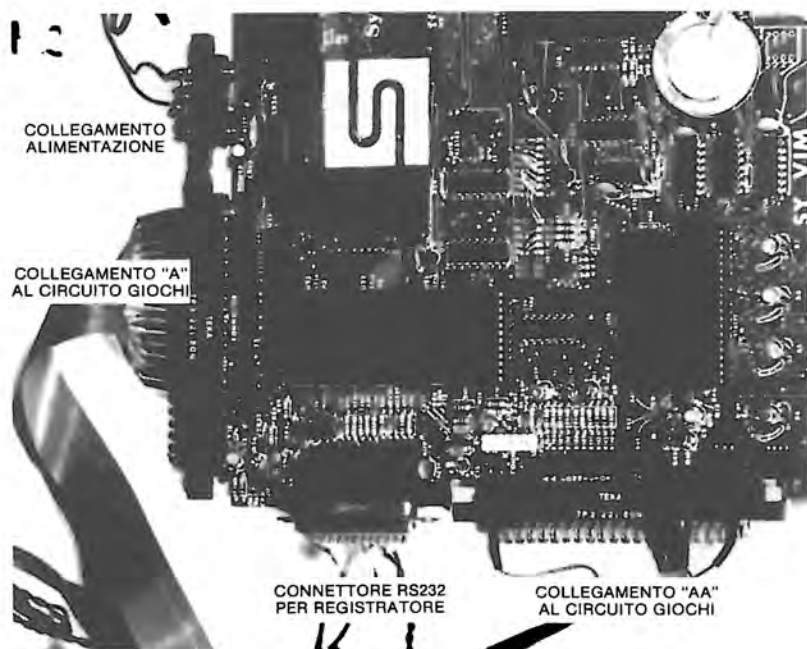


Figura 1.4: La scheda giochi è connessa al SYM con 2 connettori (notate anche i connettori di alimentazione e della cassetta)



Figura 1.5: Connessione del registratore a cassette



Figura 1.6: Il sistema è pronto per l'impiego

## INTERCONNESSIONE DELLA SCHEDA GIOCHI

### La tastiera

La Fig. 1.7 mostra i componenti della scheda. La Fig. 1.8 riporta invece la disposizione dei LED impiegati per i giochi. La tastiera impiegata in questo caso è del tipo "linea per tasto" e non impiega una disposizione a matrice. Per i giochi sono richiesti 16 tasti, anche se numerose "tastiere standard" sono dotate di più tasti, come quella impiegata nel prototipo di Fig. 1.7. Su questo prototipo, i tre tasti all'angolo destro in basso, non vengono utilizzati (si tratta dei tasti H, L e "shift" o scorrimento).

La Fig. 1.9 mostra l'impiego di un decodificatore da 1 a 16 (il 74154), impiegato per identificare il tasto premuto, con la connessione di sole quattro linee d'uscita (da PB0 a PB3) - quattro linee consentono 16 codici. Il programma di esplorazione della tastiera invierà i numeri da 0 a 15 in successione, sulle linee d'uscita PB0-PB3. In risposta, il decodificatore 74154 decodificherà i suoi ingressi (4 bit) in ciascuna delle 16 uscite, in sequenza. Per esempio, quando il numero binario "0000" viene fatto uscire sulle linee da PB0 a PB3, il 74154 collega a massa la linea 1, corrispondente al tasto "0". Questo è illustrato in Fig. 1.9. Dopo l'uscita di ogni combinazione di quattro bit, il programma di scansione legge il valore di PA7. Se il tasto correntemente collegato a massa non è stato premuto, PA7 sarà alto. Se invece il tasto corrispondente è stato premuto,

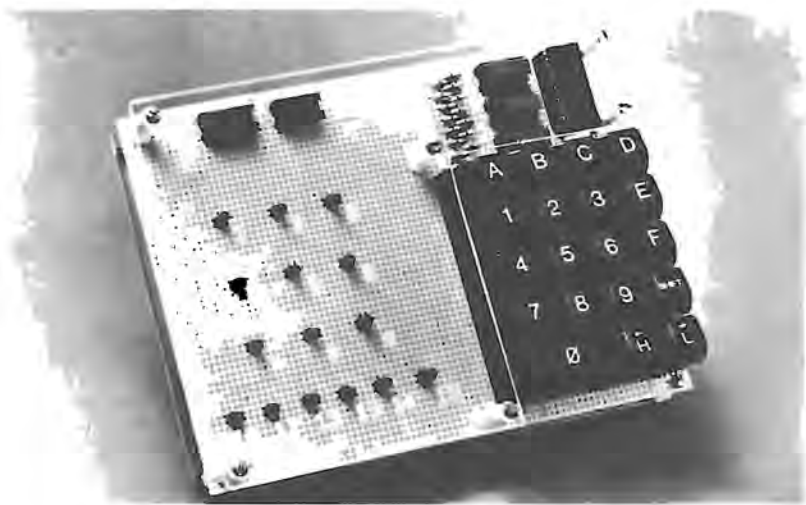


Figura 1.7: Elementi della scheda giochi (prototipo)

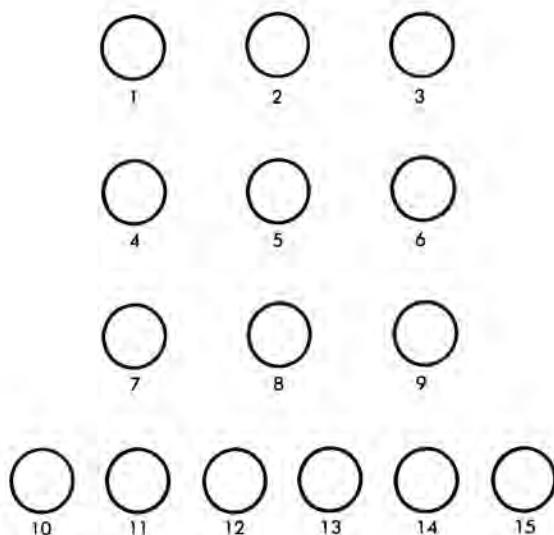


Figura 1.8: I LED

to, PA7 sarà a livello di massa e verrà letto uno "0" logico. Per esempio, in Fig. 1.10, è stata rivelata una chiusura del tasto 1. Come in qualsiasi algoritmo di scansione, un buon programma realizzerà l'eliminazione del rimbalzo della chiusura dei tasti con l'implementazione di un ritardo. Per ulteriori dettagli sulle tecniche di interfacciamento delle tastiere, si rimanda il lettore a "Tecniche di Interfacciamento dei Microprocessori" edito dal Gruppo Editoriale Jackson.

Nel progetto reale, i quattro ingressi del 74154 (da PB0 a PB3) sono connessi al VIA # 3 del SYM. PB7 è connesso allo stesso VIA. Il resistore da 3,3 K in alto a destra in Fig. 1.9, si comporta da resistore di pull-up per PA7 e garantisce un livello logico "1" finché non si verifica un collegamento a massa.

Il programma GETKEY, oppure una routine simile, viene impiegato da tutti i programmi di questo libro e viene descritto più avanti.

## I LED

La Fig. 1.11 mostra il collegamento di quindici LED. Per fornire la corrente necessaria (16 mA) vengono impiegati tre LED driver 7416.

I LED sono connessi alle linee da PA0 a PA7 e da PB0 a PB7, eccetto PB6. Queste porte appartengono al VIA # 1 del SYM. Un LED viene illuminato semplicemente selezionando l'appropriato pin d'ingresso del

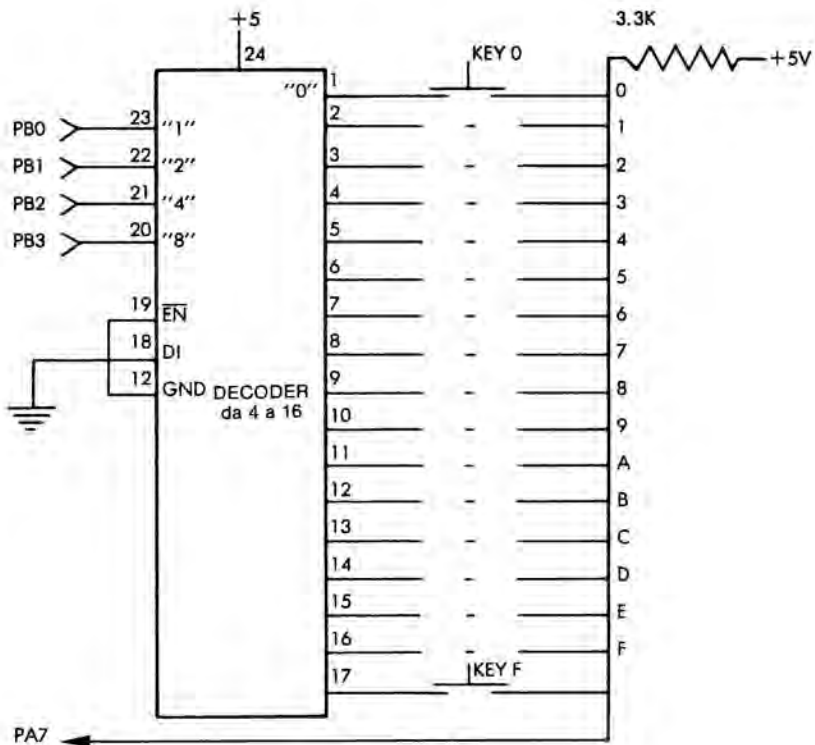


Figura 1.9: Connessione del decodificatore alla tastiera

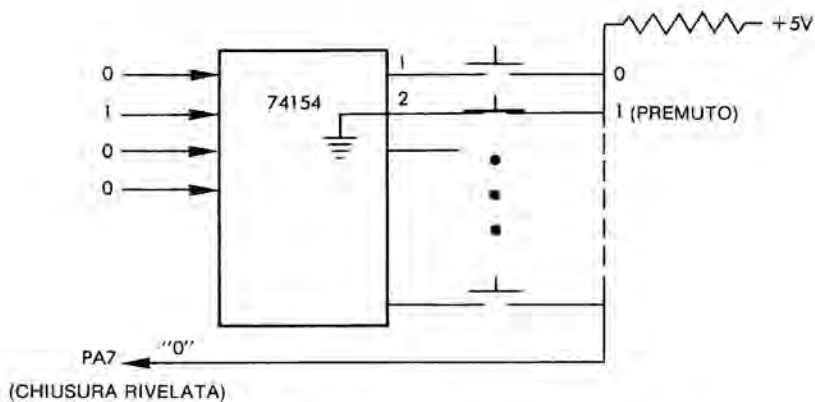


Figura 1.10: Rivelazione di una chiusura di tasto

VIA #1

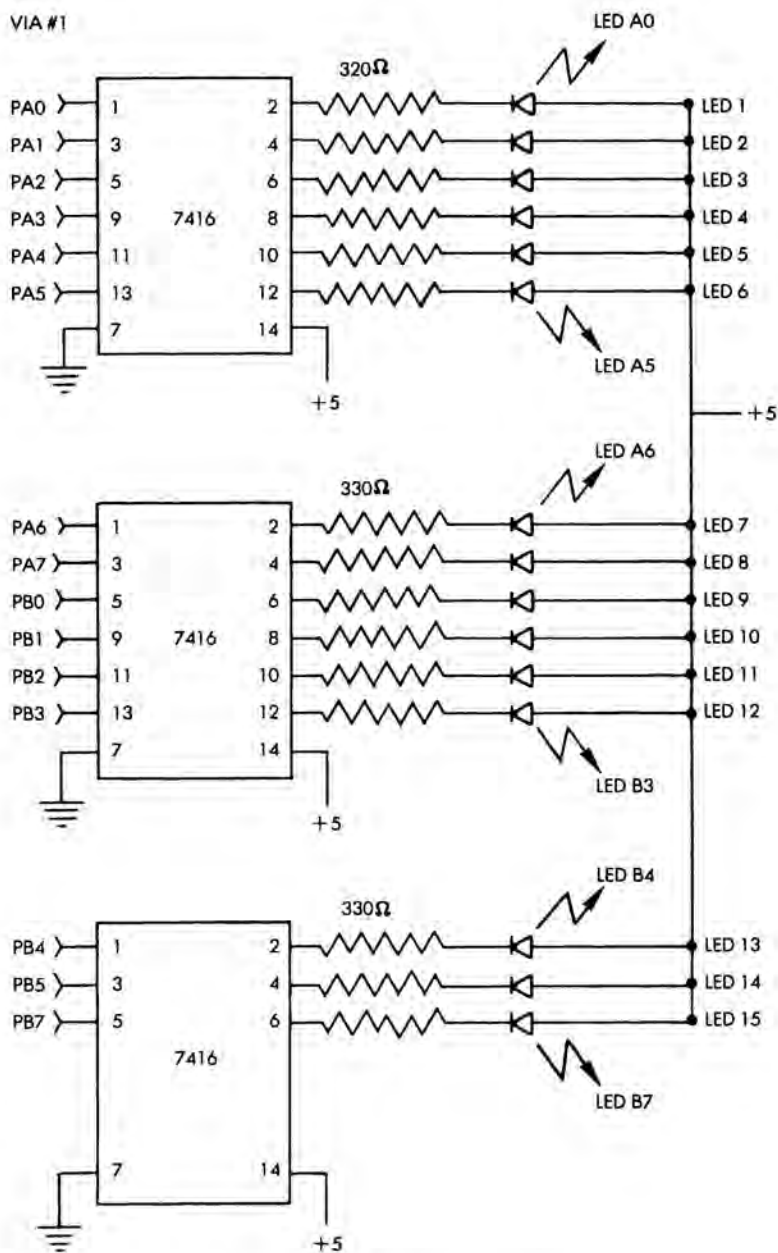


Figura 1.11: Connessione dei LED



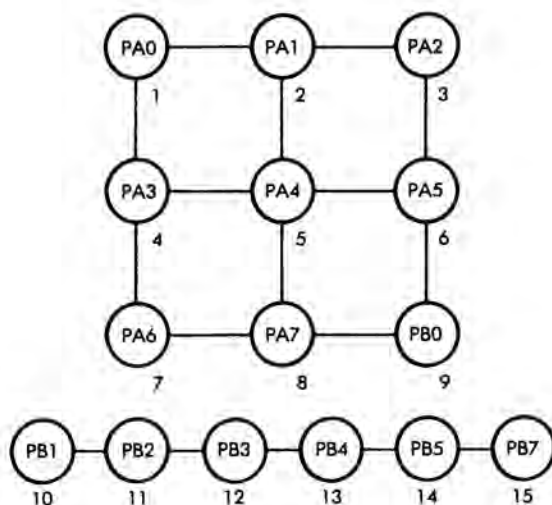


Figura 1.12: Disposizione dei LED sulla scheda

driver corrispondente. Le Fig. 1.12 ed 1.13 mostrano la conseguente disposizione.

I resistori mostrati in Fig. 1.11 sono di 330 ohm e sono progettati per limitare la corrente dei gate 7416.

Le routine d'uscita saranno descritte nel contesto dei singoli giochi.

### Componenti richiesti

- Una scheda vettore 6"x 9"
- Un decodificatore da 4 a 16 (74154)
- Tre driver invertenti sestupli (7416)
- Uno zoccolo da 24 pin
- Tre zocchi da 14 pin (per i driver)
- Una tastiera a 16 tasti, non codificata
- Quindici resistori da 330 ohm
- Un condensatore di disaccoppiamento (0,1 mF)
- Quindici LED
- Un altoparlante
- Un resistore da 50 o 110 ohm (per l'altoparlante)
- Due cavi a nastro da 16 conduttori, lunghi 4 o 5 metri
- Una scatola di montanti per terminali wire-wrap
- Conduttori wire-wrap
- Saldatore

Inoltre possono servire un saldatoio ed uno strumento per il wire-wrapping.

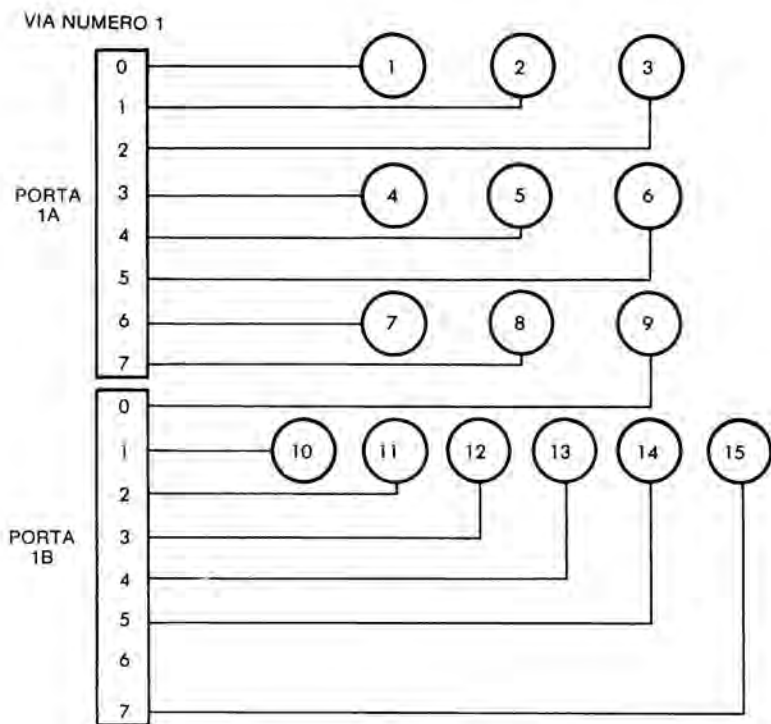


Figura 1.13: Dettagli della connessione dei LED alle porte

## Assemblaggio

Si consiglia la seguente procedura di assemblaggio: la tastiera può essere fissata direttamente sulla scheda perforata. Gli zoccoli ed i LED possono essere posizionati sulla scheda e fissati temporaneamente con nastro isolante. Tutte le connessioni possono quindi essere eseguite mediante wire-wrap. Nel prototipo le connessioni alla tastiera andranno saldate in modo da fornire una connessione affidabile, poichè esse non possono essere eseguite con il metodo wire-wrap. Per le comuni connessioni sono stati impiegati montanti per terminali wire-wrap.

Adizionalmente, sul prototipo vengono forniti due zoccoli per la connessione del cavo a nastro alla Scheda Giochi. Questi ultimi non sono indispensabili ma si consiglia vivamente il loro impiego per connettere e sconnettere efficientemente il cavo. (Essi appaiono in alto a sinistra nella Fig. 1.14). Per questo scopo sono stati impiegati due zoccoli, uno da 14 ed uno da 16 pin. Per connettere direttamente il cavo a nastro alla

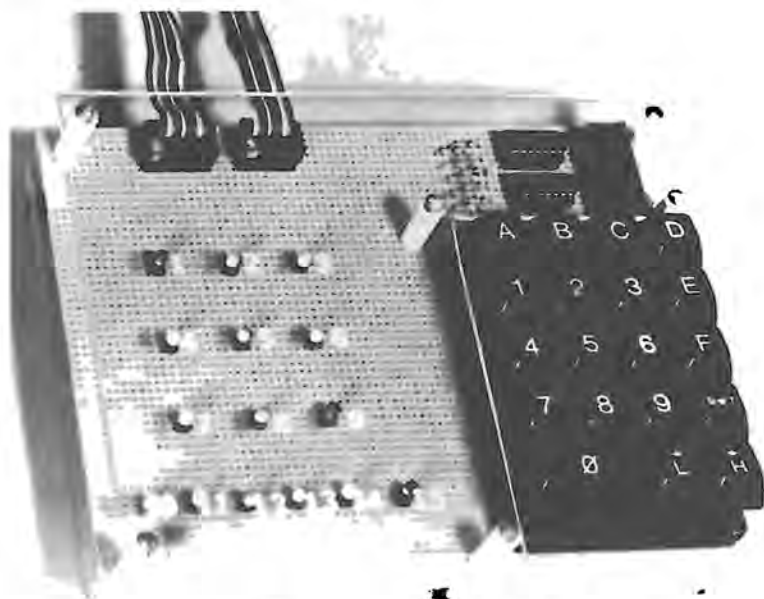


Figura 1.14: Dettagli della scheda giochi

scheda perforata si possono utilizzare, in modo alternativo, due montanti per terminali wire-wrap. L'altra estremità del cavo a nastro è semplicemente connessa ai connettori di tipo edge (bordo) del SYM. Occorre fare molta attenzione ad eseguire i collegamenti ai pin corretti di entrambe le estremità del cavo a nastro (non connettere la parte superiore in basso). La Scheda Giochi riceve l'alimentazione del SYM attraverso la connessione con cavo a nastro. La connessione del cavo in modo inverso può quindi avere effetti disastrosi.

L'altoparlante può essere connesso ad uno dei driver d'uscita PB4, PB5, PB6 oppure PB7 del VIA # 3. Ciascuna di queste porte d'uscita è equipaggiata di un transistor buffer. In serie all'altoparlante è inserita una resistenza da 110 ohm per la limitazione della corrente.

### La routine d'ingresso da tastiera

Questa routine, chiamata "GETKEY" (accetta tasto), è una utility routine (routine di utilità) che esplora la tastiera ed identifica il tasto che è stato premuto. Il codice corrispondente sarà contenuto nell'accumulatore. Essa è protetta contro il rimbalzo, ripetizione e rollover (più tasti premuti contemporaneamente).

Il rimbalzo della tastiera viene eliminato implementando un ritardo di 50 ms dalla chiusura del tasto.

Il problema della ripetizione viene risolto aspettando che il tasto correntemente premuto venga rilasciato prima che venga accettato un altro valore.

Questo corrisponde al caso in cui il tasto viene premuto per un lungo periodo di tempo. Se al caricamento della routine GETKEY un tasto è già premuto, il programma rimane inattivo finché non viene rilasciato. Quindi il programma attende la successiva chiusura di tasto. Qualora il programma di elaborazione che impiega la routine GETKEY stia eseguendo calcoli lunghi, esiste l'eventualità che l'utente possa premere un nuovo tasto prima che GETKEY sia nuovamente richiamata. Questa chiusura di tasto sarà ignorata da GETKEY e l'utente dovrà nuovamente premere il tasto.

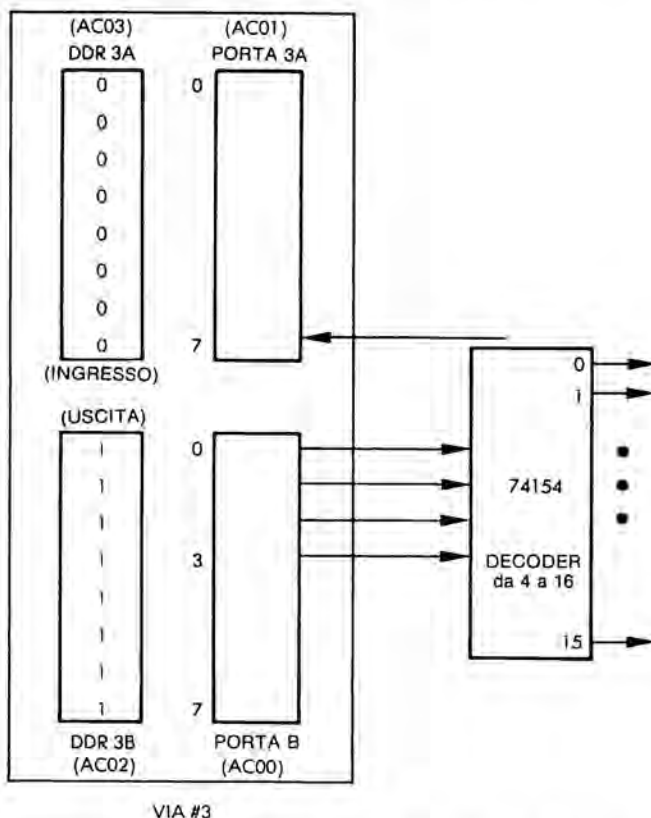


Figura 1.15: Connessione del VIA al decodificatore della tastiera

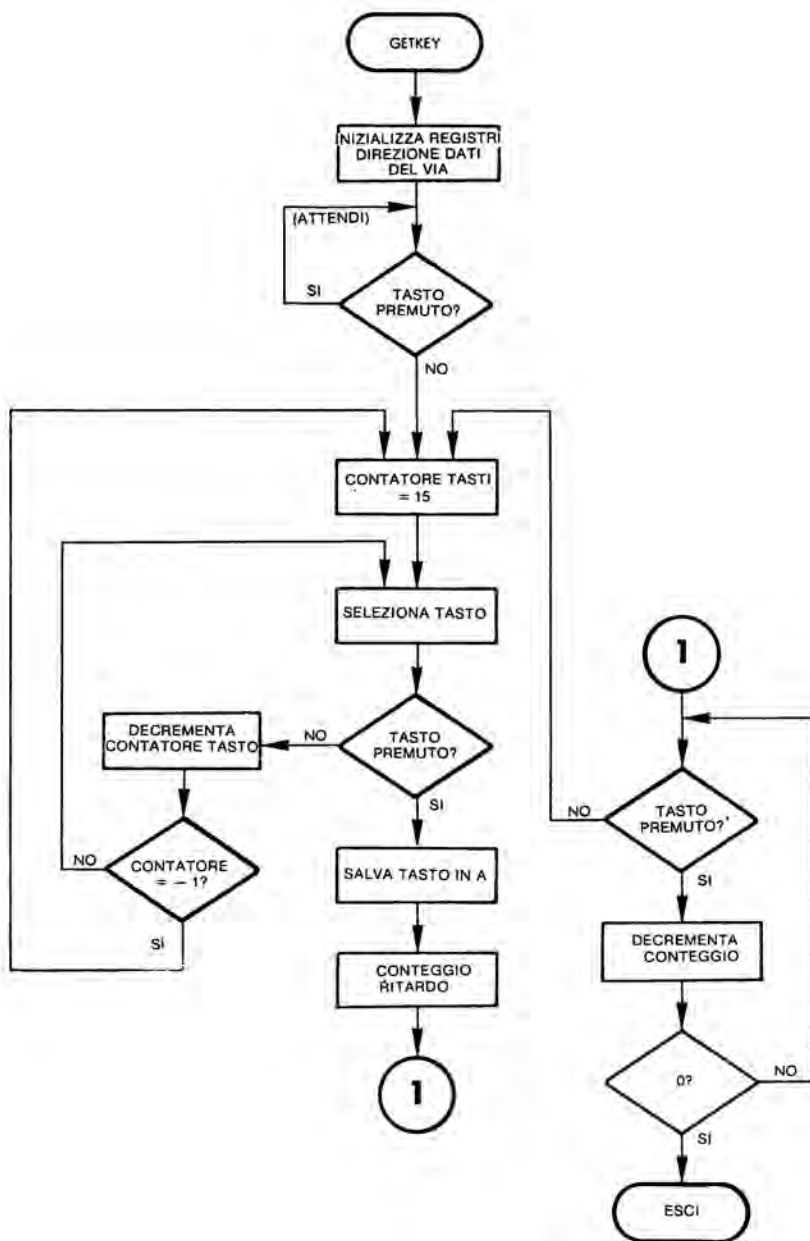


Figura 1.16: Diagramma di flusso GETKEY

La maggior parte dei programmi descritti in questo libro contengono prompt sonori sotto forma di un tono generato ogni volta che il giocatore deve rispondere. Si noti che, quando viene generato un tono, oppure durante l'esecuzione di un ciclo di ritardo, la pressione di un tasto non ha alcun effetto.

La Fig. 1.9 mostra la configurazione hardware richiesta per l'esecuzione della routine GETKEY. La Fig. 1.15 mostra il corrispondente chip d'ingresso/uscita sul SYM. Il VIA # 3 del SYM viene impiegato per comunicare con la tastiera. La porta B del VIA è configurata come uscita e le linee da 0 a 3 sono comandate dal 74154 (convertitore da 4 a 16), connesso alla tastiera stessa. La routine GETKEY farà uscire le cifre esadecimali comprese tra "0" ed "F", in sequenza sul 74154. Questo risulterà sulla messa a terra della corrispondente linea di uscita del 74154. Se viene premuto un tasto, il bit 7 del VIA # 3 della Porta A andrà a massa. Quindi la logica del programma è abbastanza semplice ed il diagramma di flusso è riportato in Fig. 1.16.

La Fig. 1.17 riporta il programma. Esaminiamolo. La routine GETKEY può essere rilocata, cioè può essere posizionata dovunque in memoria. Per ragioni di spazio essa è stata posizionata alle locazioni di memoria da 100 a 12E. È importante ricordare che questa è l'area di memoria bassa dello stack. Qualsiasi programma utente che richiede l'intero stack deve riscrivere questa routine prima di distruggerla. Per questo è posizionabile dovunque. Comunque, per tutti i programmi sviluppati in questo libro, la posizione è adatta. Le prime quattro istruzioni della routine condizionano i registri di direzione dati del VIA # 3. Il registro di direzione dati della Porta A è predisposto come ingresso (tutti zeri), mentre il registro di direzione dati per la Porta B è predisposto come uscita (tutti uni). Questo è illustrato in Fig. 1.15.

```
LDA #0
STA DDR3A
LDA #$FF
STA DDR3B
```

Per il test del bit 7 della Porta 3A sono richieste due istruzioni, che indicano se si è verificata la chiusura di un tasto:

```
START      BIT PORT3A
           BPL START
```

Il contatore dei tasti è inizialmente posto al valore 15 e sarà decrementato fino ad incontrare la chiusura di un tasto. Per contenere questo valore si impiega il registro indice X, che può essere facilmente decrementato con l'istruzione DEX:

```
RSTART     LDX #15
```

LA ROUTINE D'INGRESSO DA TASTIERA 'GETKEY'  
 LEGGE ED ELIMINA IL RIMBALZO DELLA TASTIERA,  
 RITORNA CON IL NUMERO DEL TASTO PREMUTO  
 NELL'ACCUMULATORE.  
 FUNZIONAMENTO: INVIA NUMERI DA 0 AD F AL 74154  
 (DECODER DA 4 A 16), CHE CONNETTE A MASSA UN  
 MORSETTO DEGLI SWITCH DEI TASTI, UNO ALLA VOLTA.  
 SE UN TASTO È PREMUTO, PA7 DEL VIA #3 SARA'  
 A MASSA ED IL VALORE CORRENTE APPLICATO AL  
 74154 SARA' IL NUMERO DEL TASTO. QUANDO IL  
 PROGRAMMA RIVELA UNA CHIUSURA DI TASTO CONTROLLA  
 LA CHIUSURA PER 50 MS PER ELIMINARE IL RIMBALZO.  
 NOTA: SE NESSUN TASTO È PREMUTO GETKEY È  
 IN STATO DI ATTESA.

= \$ 100

DDR3A = \$ AC03

DDR3B = \$ AC02

PORT3A = \$ AC01

PORT3B = \$ AC00

NOTA: GETKEY È NELLA PARTE BASSA  
 DELLO STACK  
 REG DI DIREZIONE DATI A PER VIA #3  
 REG DI DIREZIONE DATI B PER VIA #3  
 REGISTRI DI I/O ALLA PORTA A DEL  
 VIA #3  
 REGISTRI DI I/O ALLA PORTA B DEL  
 VIA #3

0100:	A9 00		LDA #0	
0102:	8D 03 AC		STA DDR3A	PREDISPONI PER INGRESSO LA PORTA DI STROBE DEL TASTO
0105:	A9 FF		LDA #\$FF	
0107:	8D 02 AC		STA DDR3B	PREDISPONI COME USCITA LA PORTA DEL # TASTO
010A:	2C 01 AC START	BIT	PORT3A	VEDI SE TASTO ANCORA PREMUTO ULTIMA CHIUSURA DI TASTO: STROBE DEL TASTO NEL BIT DI STATO 'N'
010D:	10 FB		BPL START	SE SI, ATTENDI IL RILASCIO DEL TASTO
010F:	A2 0F	RSTART	LDX # 15	PONI CONTATORE TASTI A 15
0111:	8E 00 AC	NXTKEY	STX PORT3B	USCITA # TASTO A 74154
0114:	2C 01 AC	BIT	PORT3A	VEDI SE TASTO PREMUTO: STROBE IN 'N'
0117:	10 05		BPL BOUNCE	SE SI, ELIMINA RIMBALZO
0119:	CA		DEX	DECREMENTA # TASTO
011A:	10 F5		BPL NXTKEY	NO, TASTO SUCCESSIVO
011C:	30 F1		BMI RSTART	SCAVALCA
011E:	8A	BOUNCE	TXA	SALVA IL NUMERO DEL TASTO IN A
011F:	A0 12		LDY # \$ 12	CARICA CONT CICLO ESTERNO PER RITARDO DI 50 MS.
0121:	A2 FF	LP1	LDX # \$ FF	CICLO INTERNO DI 11 MICROSEC.
0123:	2C 01 AC	LP2	BIT PORT3A	VEDI SE TASTO ANCORA PREMUTO
0126:	30 E7		BMI RSTART	SE NO, TASTO NON VALIDO, RESTART
0128:	CA		DEX	
0129:	D0 FB		BNE LP2	QUESTO CICLO DURA 2115*5 MICROSEC.
012B:	88		DEY	
012C:	D0 F3		BNE LP1	CICLO ESTERNO: IL TOTALE È 50 MS.
012E:	60		RTS	FATTO: # TASTO IN A.

# SYMBOL TABLE:

DDR3A	AC03	DDR3B	AC02	PORT3A	AC01
PORT3B	AC00	START	010A	RSTART	010F
NXTKEY	0111	BOUNCE	011E	LP1	0121
LP2	0123				
DONE					

Figura 1.17: Programma GETKEY



Questo valore (15) viene quindi fatto uscire sul 74154 e genera la scelta della linea 17 connessa al tasto 15 ("F"). La precedente istruzione BIT viene impiegata per provare la condizione del bit 7 della Porta 3A, in modo da determinare se questo tasto è stato premuto.

```
NXTKEY    STX PORT3B
           BIT PORT3A
           BPL BOUNCE
```

Se il tasto è stato premuto si verifica una ramificazione a "BOUNCE" (rimbalzo), e deve essere implementato un ritardo per eliminare il rimbalzo; in caso contrario viene decrementato il contatore e si ha il test per underflow (superamento del limite inferiore). Prima che il contatore diventi negativo si verifica una ramificazione alla locazione NXTKEY. Questo ciclo viene ripetuto finchè si trova un tasto premuto oppure il contatore diviene negativo. In questo caso la routine ritorna alla locazione RSTART, facendo ripartire il processo:

```
DEX
BPL NXTKEY
BMI RSTART
```

Si noti che, nel caso della pressione contemporanea di più tasti, rivelerà il tasto premuto di valore più elevato. In altre parole, se sono stati premuti contemporaneamente i tasti "F" e "3", verrà identificato come premuto il tasto "F", mentre il tasto "3" verrà ignorato. Per evitare questo problema occorre introdurre la *protezione contro rollover multipli* che viene suggerita come esercizio:

**Esercizio 1-1:** *Per evitare il rollover di tasti multipli, si modifichi la routine GETKEY in modo che vengano ispezionate tutte le chiusure dei 15 tasti. Se viene premuto più di un tasto, la chiusura viene ignorata fino a che non si rivela la chiusura di un solo tasto.*

Una volta identificata la chiusura di un tasto, il numero corrispondente al tasto viene salvato nell'accumulatore. Viene quindi implementato un ciclo di ritardo per fornire un tempo di eliminazione rimbalzo di 50 ms. Durante questo ciclo, la chiusura del tasto viene costantemente osservata. Il ritardo viene implementato con l'impiego di una tecnica a ciclo annidato di due livelli.

```
BOUNCE    TXA
           LDY #$12
LP1        LDX #$FF
LP2        BIT PORT3A
```

BMI RSTART  
DEX  
BNE LP2  
DEY  
BNE LP1

**Esercizio 1-2:** Il valore impiegato per il contatore del ciclo (loop) esterno può non essere sufficientemente preciso ("12" oppure 12 esadecimale). Calcolate la durata esatta del ritardo implementato dalle precedenti istruzioni, impiegando le tabelle che mostrano la durata di ogni istruzione (Appendice).

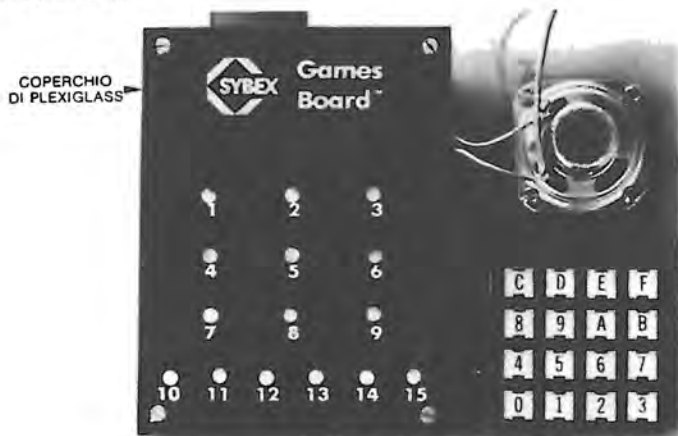


Figura 1.18: Scheda giochi in produzione

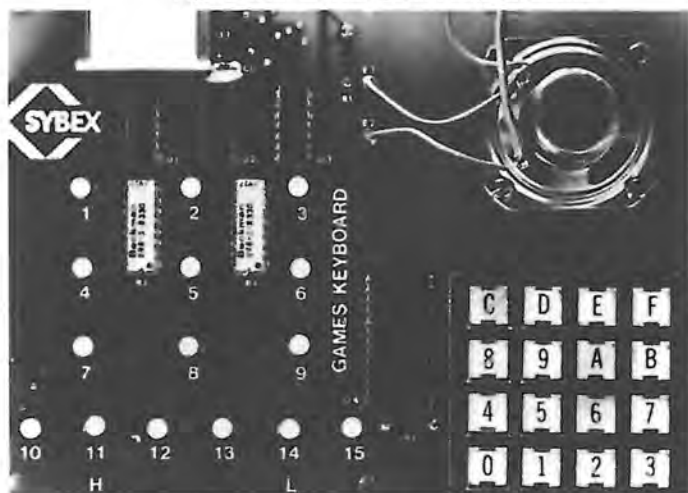


Figura 1.19: Rimozione del coperchio

## SOMMARIO

L'esecuzione dei programmi dei giochi richiede una semplice Scheda Giochi che fornisce le prestazioni d'ingresso/uscita di base. In questo capitolo è stata descritta l'interfaccia hardware e software richiesta. Le Fig. 1.18 ed 1.19 mostrano la scheda assemblata che deriva dal precedente prototipo.

## CAPITOLO 2

# GIOCO MUSICALE

### LE REGOLE

Questo gioco consente di generare musica direttamente dalla tastiera del computer. Inoltre il programma registra contemporaneamente le note emesse e su richiesta può rieseguirle. I tasti da "0" a "C" sulla tastiera vengono impiegati per generare le note musicali. (Vedere Fig. 2.1). Il tasto "D" viene impiegato per specificare una pausa. Il tasto "E" viene impiegato per rieseguire la sequenza musicale immagazzinata in memoria. Infine il tasto "F" viene impiegato per azzerare la memoria, cioè per iniziare un nuovo gioco. Il paragrafo seguente descrive la normale sequenza del gioco.

### UNA GIOCATA TIPICA

Premete il tasto "F" per iniziare un nuovo gioco. L'emissione di un suono di tre note confermerà che la memoria interna è stata cancellata.

A (A)	B (B)	C (C)	D (REST)
1 (A)	2 (B)	3 (C)	E (PBK)
4 (D)	5 (E)	6 (F)	F (RST)
7 (F#)	8 (G)	9 (G#)	0 (G)

NUMERO TASTI	NOTA	NUMERO TASTI	NOTA
0	G	8	G
1	A	9	G#
2	B	A	A
3	C	B	B
4	D	C	C
5	E	D	REST
6	F	E	PLAY BACK
7	F#	F	RESTART

Figura 2.1: Esecuzione di musica da tastiera

**Nona sinfonia**

5-5-6-8-8-6-5-4-3-3-4-5-5-4-4-D-5-  
 5-6-8-8-6-5-4-3-3-4-5-4-3-3-D-4-4-  
 5-3-4-6-5-3-4-6-5-4-3-4-D

**Clementine:**

3-3-3-D-2-D-5-5-5-D-3-D-3-5-8-D-D-  
 8-6-5-4-D-D-D-4-5-6-D-6-D-5-4-5-D-  
 3-D-3-5-4-D-D-2-3-4-3

**Frere Jacques:**

3-4-5-3-3-4-5-3-5-6-8-D-5-6-8-D-8-  
 A-8-6-5-D-3-D-8-A-8-6-5-D-3-D-3-D-  
 2-D-3-D-D-D-3-D-2-D-3

**Jingle Bells:**

5-5-5-D-5-5-5-D-5-8-3-4-5-D-D-D-6-  
 6-6-6-6-5-5-5-8-8-6-4-3

**London Bridge:**

8-A-8-6-5-6-8-D-4-5-6-D-5-6-8-D-8-  
 A-8-6-5-6-8-D-4-D-8-D-5-3

**Mary Had a Little Lamb:**

5-4-3-4-5-5-5-D-4-4-4-D-5-8-8-D-5-  
 4-3-4-5-5-5-5-4-4-5-4-3

**Row Row Row Your Boat:**

3-D-3-D-3-4-5-D-5-4-5-6-8-D-D-D-C-  
 C-8-8-5-5-3-3-8-6-5-4-3

**Silent Night:**

8-D-D-A-8-D-5-D-D-D-8-D-D-A-8-D-5-  
 D-D-D-3-D-D-3-D-8-D-D-D-C-D-D-C-  
 D-8-D-D-C-D-8-5-8-D-6-D-4-D-3

**Twinkle Twinkle Little Star:**

3-3-8-8-A-A-8-D-6-6-5-5-4-4-3-D-8-  
 8-6-6-5-5-4-D-3-3-8-8-A-A-8-D-6-6-  
 5-5-4-4-3

Figura 2.2: Toni semplici per la musica del computer

Eseguite l'accordo sui tasti da "0" a "D" (impiegando note e pause). È possibile eseguire e memorizzare fino a 254 note. In qualsiasi istante può essere premuto il tasto di riproduzione ("E") e quindi le note e pause che sono state impostate sulla tastiera (e contemporaneamente memorizzate) saranno riprodotte. La sequenza musicale può essere riprodotta indefinitamente semplicemente premendo il tasto "E". La Fig. 2.2 mostra alcuni esempi di semplici accordi o sequenze musicali che possono essere eseguite sul computer.

## LE CONNESSIONI

Questo gioco impiega la tastiera e l'altoparlante. L'altoparlante è connesso in serie alle linee d'uscita dotate di buffer della Porta B del VIA #3, attraverso una resistenza di limitazione della corrente da 110 ohm. Vengono utilizzate le linee PB4, PB5, PB6 o PB7 del VIA # 3, che sono pilotate da un transistor buffer sul SYM'. Per ottenere musica di elevata qualità si consiglia di installare l'altoparlante in un cassa. Inoltre si può regolare il valore del resistore per agire sul volume (senza scendere al di sotto dei 50 ohm) per limitare la corrente nel transistor.

## L'ALGORITMO

Un tono (nota) viene generato semplicemente inviando un'onda quadra di frequenza opportuna all'altoparlante, cioè commutandolo on ed off alla frequenza richiesta. Questo è illustrato in Fig. 2.3. L'intervallo di tempo in cui l'altoparlante è on oppure off è detto semi-periodo. In questo programma è disponibile la gamma di frequenza da 195 a 253 Hertz. Se  $N$  è la frequenza, il periodo  $T$  è l'inverso della frequenza, cioè:

$$T = 1/N$$

Quindi il semi-periodo va da  $1/(2 \times 195) = 0,002564$  a  $1/(2 \times 523) = 0,000956$  secondi. Per implementare la frequenza richiesta verrà impiegato un ciclo di ritardo classico.

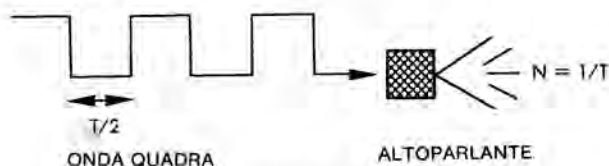


Figura 2.3: Generazione di un tono

Di seguito vengono presentati i calcoli reali dei vari parametri del programma.

## IL PROGRAMMA

Il programma è localizzato agli indirizzi di memoria da 200 a 2DD e la sequenza musicale registrata, o l'accordo, è memorizzata a partire dalla locazione di memoria 300. In 127 byte possono essere registrate fino a 254 note.

### Strutture dati

In questo programma vengono impiegate tre tabelle. Esse sono riportate in Fig. 2.4. L'accordo registrato è memorizzato in una tabella che inizia all'indirizzo 300. Le costanti delle note, impiegate per determinare la frequenza di eccitazione dell'altoparlante, sono memorizzate in una tabella di 16 byte localizzata all'indirizzo di memoria 2C4. La durata delle note, cioè il numero di semi-periodi richiesti per implementare una durata uniforme della nota, pari a circa 0,21 secondi, è memorizzata in una tabella di 16 byte che inizia all'indirizzo di memoria 2D1. All'interno della tabella dell'accordo sono impiegati dei puntatori di due "nibble" (gruppo di 4 bit): PILEN durante l'ingresso a PTR durante l'uscita. (Ogni byte di 8 bit di questa tabella contiene due note). Per ottenere l'ingresso effettivo alla tabella dal puntatore di nibble, il puntatore viene semplicemente fatto scorrere di una posizione a destra. Il valore che rimane diventa il puntatore di byte, mentre il bit che è entrato nel flag carry (riporto) specifica la metà sinistra o destra del byte. Le due tabelle chiamate COSTANTI e DURATE DELLE NOTE sono semplicemente tabelle di riferimento impiegate per determinare la semi-frequenza di una nota ed il numero di volte di attivazione dell'altoparlante una volta che è stata identificata e specificata una nota. L'accesso ad entrambe queste tabelle è indiretto con l'impiego del registro X.

### Elementi di teoria della musica

Prima di procedere alla descrizione del programma reale, conviene richiamare brevemente le convenzioni generali della musica. Le frequenze impiegate per generare le note richieste sono derivate da una scala uniforme in cui le frequenze di note successive stanno nel rapporto:

$$1 : \sqrt[12]{2}$$

La Fig. 2.5 fornisce le frequenze centrali C di ottava. Il calcolo delle



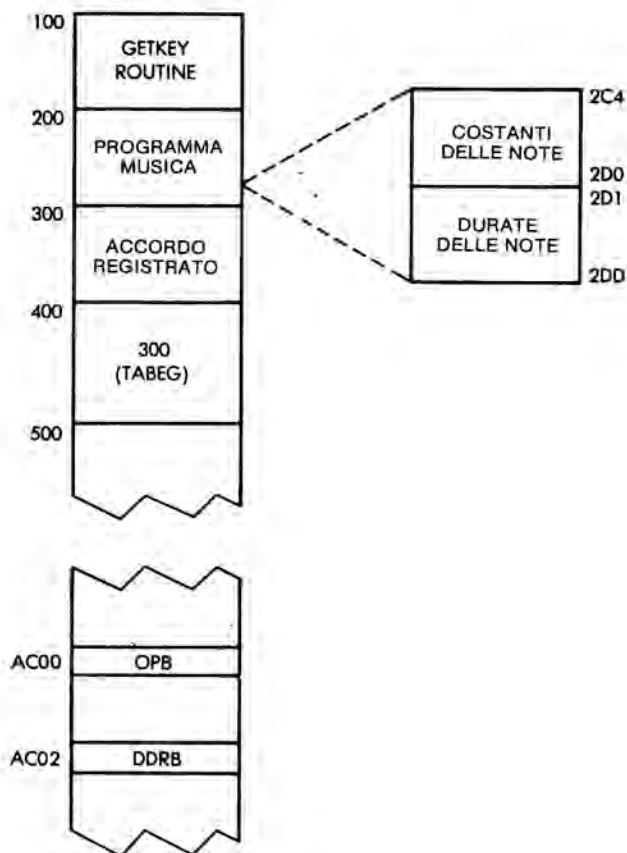


Figura 2.4: Mappa di memoria

corrispondenti frequenze dell'ottava più alta o più bassa si esegue semplicemente moltiplicando o dividendo per due, rispettivamente.

### Generazione del tono

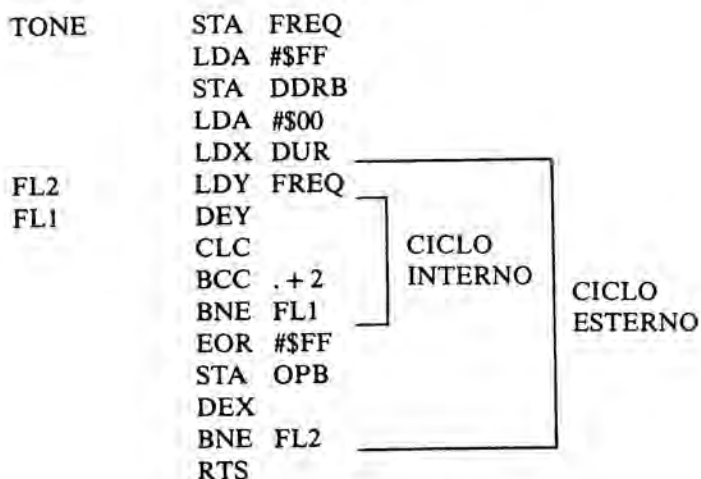
Il semiperiodo di ritardo dell'onda quadra inviata all'altoparlante viene implementato con l'impiego di un ciclo di programma avente un tempo di ciclo di base di  $10 \mu s$ . Nel programma, l'"indice del ciclo", o contatore di iterazione, viene impiegato per contare il numero di cicli di  $10 \mu s$  eseguiti. Il ciclo genererà un ritardo totale pari a:

$$(\text{indice del ciclo}) \times 10 - 1 \text{ microsecondi}$$

NOTE	FREQUENZA (HERTZ)
A	220.00
A#	223.08
B	246.94
C	261.62
C#	277.18
D	293.66
D#	311.13
E	329.63
F	349.23
F#	369.99
G	391.99
G#	415.30

Figura 2.5: Frequenze di centro ottava C

All'ultima iterazione del ciclo (quando l'indice del ciclo è decrementato a zero), interviene la ramificazione. Questa istruzione di ramificazione sarà eseguita più velocemente, cosicchè occorre sottrarre un microsecondo (supponendo un clock di 1 MHz) dalla durata totale del ritardo. La routine per la generazione del tono è la seguente:



Si noti il classico progetto a cicli annidati. Il ciclo esterno, ogni volta che viene eseguito, aggiunge un ritardo di 13 microsecondi: 14 microsecondi

per le istruzioni extra (LDY, EOR, STA, DEX e BNE), meno un microsecondo per compensare la ramificazione che non si è verificata nel ciclo più interno. Quindi il ritardo totale introdotto da ciclo più esterno vale:

$$(\text{indice del ciclo}) \times 10 + 13 \text{ microsecondi}$$

Si ricordi che un attraversamento del ciclo più esterno rappresenta soltanto un semiperiodo della nota.

### Calcolo delle costanti della nota

Sia "ID" il ritardo introdotto dal ciclo interno ed "OD" il ritardo addizionale introdotto dal ciclo esterno. Nel paragrafo precedente si è stabilito che il semi-periodo è  $T/2 = (\text{indice del ciclo}) \times 10 + 13$  ovvero

$$T/2 = (\text{indice del ciclo}) \times ID + OD$$

La costante della nota che viene memorizzata nella tabella è il valore dell'indice richiesto dal programma. Dall'equazione si può ricavare facilmente che:

$$\text{costante della nota} = \text{indice del ciclo} = (T - 2 \times OD)/2 \times ID$$

Il periodo può essere espresso in funzione della frequenza come  $T = 1/N$  ovvero, in microsecondi:

$$T = 10^6/N$$

Infine la precedente equazione diventa:

$$\text{costante della nota} = (10^6/N - 2 \times OD)/2 \times ID$$

Per esempio, calcoliamo la costante della nota corrispondente alla frequenza centrale C. La fig. 2.5 riporta la frequenza corrispondente al centro C: vale 261,62 Hertz. Si è visto precedentemente che il ritardo "OD" vale 13 microsecondi, mentre "ID" è stato scelto pari a 10 microsecondi. L'equazione della costante della nota diventa allora:

$$\begin{aligned} \text{costante della nota} &= \frac{(10^6/N - 2 \times 13)/2 \times 10}{1000000/261,62 - 26} \\ &= \frac{20}{20} \\ &= 190 \text{ (oppure BE in esadecimale)} \end{aligned}$$

Si può facilmente verificare che questo corrisponde al quarto ingresso

della tabella all'indirizzo NOTAB (vedere Fig. 2.9 alla fine del listing, all'indirizzo 02C4). La Fig. 2.6 mostra le costanti della nota.

NOTA		NOTA	COSTANTE	NOTA	COSTANTE	
SOTTO CENTRO C	G FE A E2 B C9	CENTRO C	C	BE	SOPRA CENTRO C	C 5E
			D	A9		
			E	96		
			F	8E		
			F#	86		
			G	7E		
			G#	77		
			A	70		
			B	64		

Figura 2.6: Costanti delle note

**Esercizio 2-1:** Impiegando la tabella della Fig. 2.6, calcolate la frequenza corrispondente, e controllate se le costanti sono state scelte correttamente.

### Calcolo delle durate delle note

La tabella DURTAB memorizza le durate delle note espresse in numeri equivalenti al numero di semi-periodi di ciascuna nota. Queste durate sono state calcolate per implementare una durata uniforme di circa 0,2175 secondi per nota. Se D è la durata e T il periodo, risulta:

$$D \times T = 0,2175$$

dove D è espresso come numero di periodi. Poichè, in pratica, sono impiegati i semi-periodi, il numero richiesto di semi-periodi è:

$$D' = 2D = 2 \times 0,2175 \times N$$

Per esempio, nel caso del centro C:

$$D = 2 \times 0,2175 \times 261,62 = 133,8 \approx 114 \text{ decimale (oppure 72 esadecimale)}$$

**Esercizio 2-2:** Calcolare le durate delle note impiegando l'equazione precedente e la tabella di frequenze della Fig. 2.5 (che richiede di essere espansa). Si verifichi inoltre che ci sia accordo con i numeri della tabella DURATAB all'indirizzo 2D1. (Vedere Fig. 2.9).

## Implementazione del Programma

Il programma è stato strutturato in due parti logiche. La Fig. 2.7 riporta il corrispondente diagramma di flusso. La prima parte del programma è responsabile della raccolta delle note e dell'inizio alla label "NUMKEY". (Il programma appare in Fig. 2.9). La seconda parte inizia alla label "PLAYEM" ed ha la funzione di eseguire le note memorizzate. Entrambe le parti del programma impiegano la subroutine PLAYNOTE che preleva le costanti delle note e delle durate, ed esegue la nota. Questa routine inizia alla label "PLAYIT" ed il relativo diagramma di flusso è riportato in Fig. 2.8.

Le principali routine si chiamano rispettivamente NXKEY, NUMKEY e BEEP3 per il programma di raccolta delle note e PLAYEM e DELAY per il programma di esecuzione delle note. Infine le routine di utilità comune sono TONE e PLAYIT.

Esaminiamo in dettaglio queste routine. Il programma risiede agli indirizzi di memoria da 200 in poi. Si noti che il programma, come la maggior parte di quelli di questo libro, assume la disponibilità della routine GETKEY descritta al Capitolo 1.

Il modo di operare della routine GETKEY è immediato. La successiva chiusura di tasto è ottenuta chiamando la routine GETKEY:

```
START      LDA #0
            STA PILEN   Inizializza la lunghezza della lista a 0
            CLC
NXKEY      JSR GETKEY
```

Il valore letto viene quindi confrontato con le costanti "15" e "14" per l'azione speciale. Se non c'è uguaglianza, la costante viene memorizzata nella lista delle note impiegando la routine NUMKEY.

```
            CMP #15
            BNE NXTST
            JSR BEEP3
            BCC START
NXTST      CMP #14
            BNE NUMKEY
            JSR PLAYEM
            CLC
            BCC NXKEY
```

**Esercizio 2-3:** *Perchè si utilizzano le ultime due istruzioni di questa routine invece di un salto incondizionato? Quali sono i vantaggi e gli svantaggi di questa tecnica?*

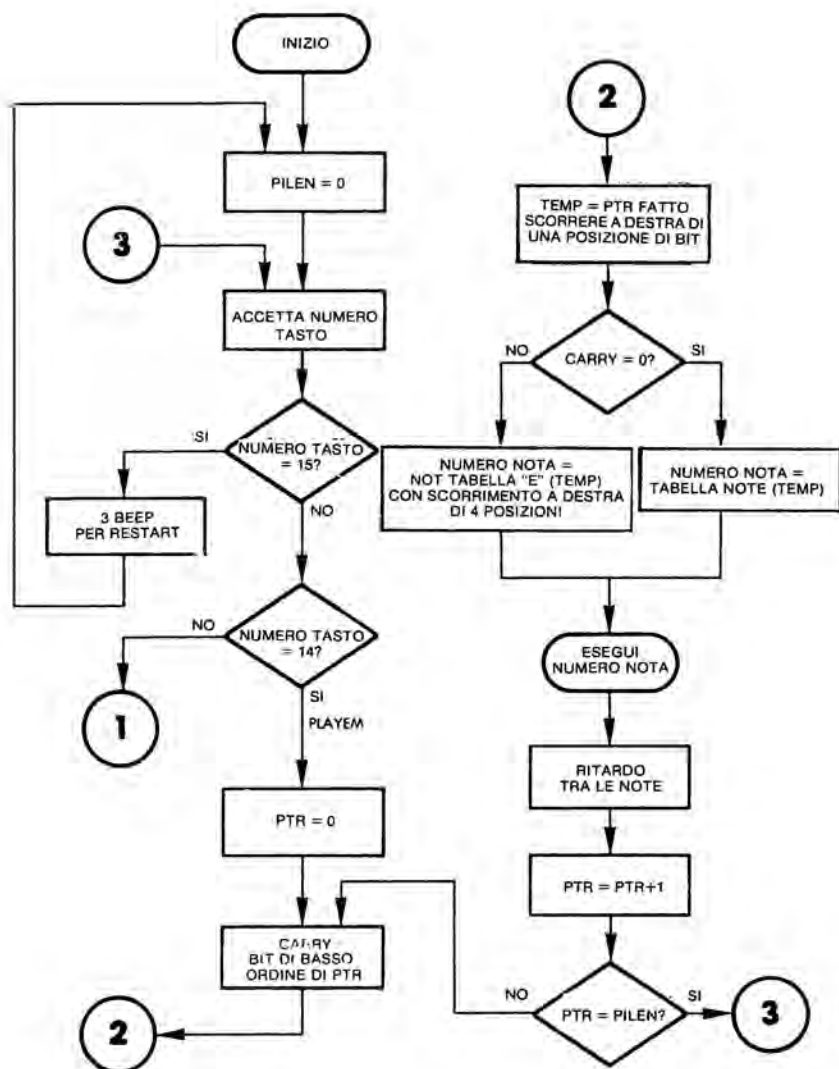


Figura 2.7: Diagramma di flusso per la generazione della musica

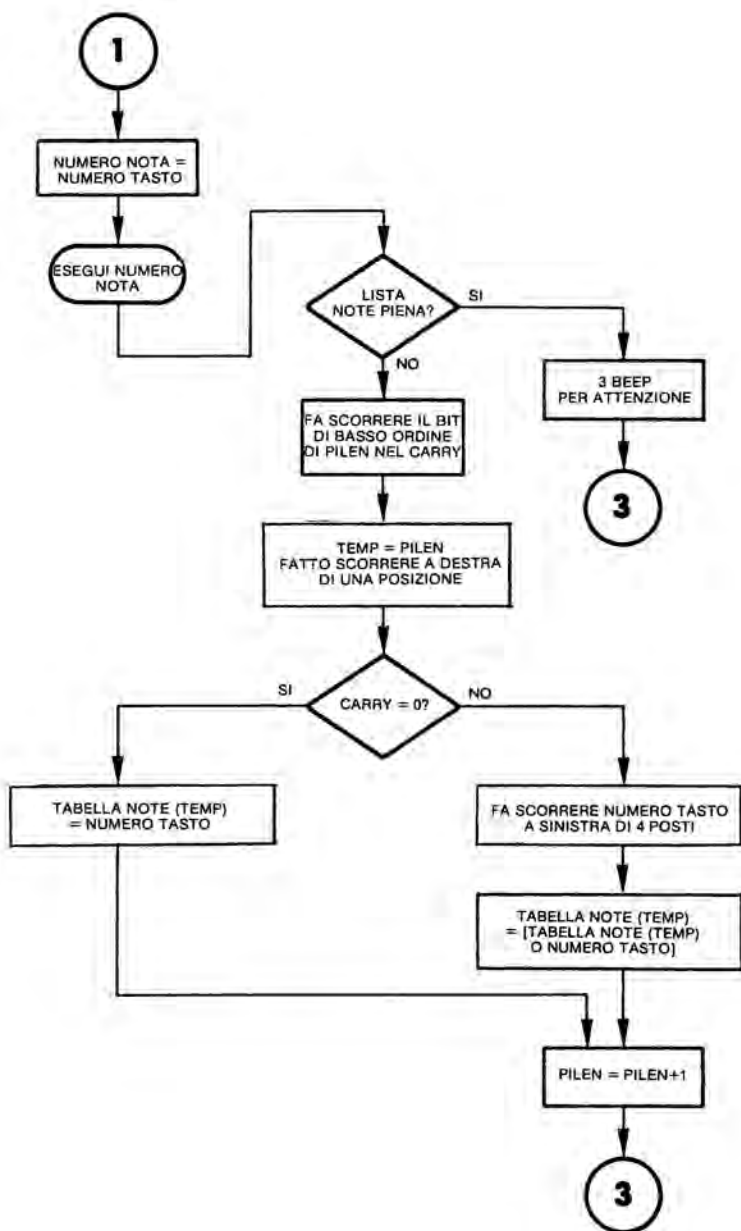


Figura 2.7 Diagramma di flusso per la generazione della musica (continua)



Figura 2.8: Diagramma di flusso PLAYIT



```

PROGRAMMA GIOCO MUSICALE
IMPIEGA UNA TASTIERA DI 16 TASTI ED UN ALTOPARLANTE
BUFFERED. IL PROGRAMMA ESEGUE LE NOTE MUSICALI
MEMORIZZATE. ESISTONO DUE MODI DI FUNZIONAMENTO:
INGRESSO ED ESECUZIONE. IL MODO D'INGRESSO
VIENE ESCLUSO E TUTTI I TASTI PREMUTI (0-D) SONO
MEMORIZZATI PER LA RIESECUZIONE. SE SI VERIFICA
UN OVERFLOW, L'UTENTE VIENE AVVERTITO CON UN
SEGNALE DI TRE TONI. LO STESSO TONO VARIABILE
VIENE IMPIEGATO ANCHE PER SEGNALARE UN RESTART
DEL PROGRAMMA.

GETKEY = $ 100
PILEN = $ 00 ; LUNGHEZZA DELLA LISTA DELLE NOTE
TEMP = $ 01 ; MEMORIZZAZIONE TEMPORANEA
PTR = $ 02 ; LOCAZIONE CORRENTE NELLA LISTA
FREQ = $ 03 ; MEMORIZZAZIONE TEMPORANEA DELLA
; FREQUENZA
DUR = $ 04 ; MEMORIZZAZIONE TEMPORANEA DELLA
; DURATA
TABEG = $ 300 ; TABELLA DELLA MUSICA MEMORIZZATA
OPB = $ AC00 ; PORTA D'USCITA B DEL VIA
DDRB = $ AC02 ; REGISTRO DIREZIONE DELLA PORTA B
; DEL VIA
; = $ 200 ; ORIGINE

; INTERPRETE LINEA COMANDO
; $F COME INGRESSO SIGNIFICA RESET PUNTATORI, RESTART.
; $E SIGNIFICA NOTE MEMORIZZATE CORRENTEMENTE ESEGUITE
; NIENT'ALTRO È MEMORIZZATO PER LA RIESECUZIONE

0200: A9 00 ; START LDA # 0 ; AZZERA LA LUNGHEZZA DELLA LISTA
; DELLE NOTE
0202: 85 00 STA PILEN
0204: 18 CLC ; AZZERA L'INDICATORE DI NIBBLE
0205: 20 00 01 NXKEY JSR GETKEY
0208: C9 0F CMP # 15 ; È IL TASTO # 15?
020A: D0 05 BNE NXTST ; NO, ESEGUI UN ALTRO TEST
020C: 20 87 02 JSR BEEP3 ; AVVERTI L'UTENTE PER L'AZZERAMENTO
020F: 90 EF BCC START ; AZZERA I PUNTATORI ED INIZIA DI NUOVO
0211: C9 0E NXTST CMP # 14 ; È IL TASTO # 14?
0213: D0 06 BNE NUMKEY ; NO, IL TASTO È IL NUMERO DELLE NOTE
0215: 20 48 02 JSR PLAYEM ; ESEGUI LE NOTE
0218: 18 CLC
0219: 90 EA BCC NXKEY ; ACCETTA IL COMANDO SUCCESSIVO

; ROUTINE PER CARICARE LA LISTA DELLE NOTE CON LE NOTE
021B: 85 01 NUMKEY STA TEMP ; SALVA TASTO, LIBERA A
021D: 20 70 02 JSR PLAYIT ; ESEGUI LA NOTA
0220: A5 00 LDA PILEN ; ACCETTA LA LUNGHEZZA DELLA LISTA
0222: C9 FF CMP # $ FF ; OVERFLOW?
0224: D0 05 BNE OK ; NO, AGGIUNGI LA NOTA ALLA LISTA
0226: 20 87 02 JSR BEEP3 ; SÌ, AVVERTI L'UTENTE
0229: 90 DA BCC NXKEY ; RITORNA AL MODO D'INGRESSO
022B: 4A OK LSR A ; FA SCORRERE IL BIT BASSO NEL
; PUNTATORE DEL NIBBLE
022C: A8 TAY ; IMPIEGA IL PUNTATORE DEL NIBBLE
; FATTO SCORRERE
; COME REGISTRO INDICE
022D: A5 01 LDA TEMP ; RIPRISTINA # TASTO
022F: B0 09 BCS FINBYT ; SE IL BYTE HA GIÀ UN NIBBLE,
; FINISCILO E MEMORIZZALO

```

Fig. 2.9: Programma per la generazione della musica

```

0231: 29 0F          AND # %00001111; PRIMO NIBBLE, MASCHERA IL NIBBLE
                                ; ALTO
0233: 99 00 03      STA TABEG,Y    ; SALVA IL 1/2 BYTE NON TERMINATO
0236: E6 00          INC PILEN      ; PUNTA AL NIBBLE SUCCESSIVO
0238: 90 CB          BCC NXKEY      ; ACCETTA LA PRESSIONE DI TASTO
                                ; SUCCESSIVA
023A: 0A           FINBYT ASL A      ; FA SCORRERE IL NIBBLE 2 ALL'ORDINE
                                ; ELEVATO
023B: 0A           ASL A
023C: 0A           ASL A
023D: 0A           ASL A
023E: 19 00 03      ORA TABEG,Y    ; UNISCI 2 NIBBLE IN UN BYTE
0241: 99 00 03      STA TABEG,Y    ; ... E MEMORIZZALO
0244: E6 00          INC PILEN      ; PUNTA AL NIBBLE SUCCESSIVO NEL BYTE
                                ; SUCCESSIVO
0246: 90 BD          BCC NXKEY      ; RITORNO

; ROUTINE PER L'ESECUZIONE DELLE NOTE

0248: A2 00      PLAYEM LDX # 0      ; AZZERA PUNTATORE
024A: 86 02      STX PTR
024C: A5 02      LDA PTR              ; CARICA ACCUM CON IL VALORE
                                ; DEL PUNTATORE CORRENTE
024E: 4A      LOOP LSR A              ; SCORRIMENTO DELL'INDICATORE
                                ; NIBBLE NEL CARRY
024F: AA          TAX                ; USA IL PUNTATORE DEL NIBBLE FATTO
                                ; SCORRERE
                                ; COME PUNTATORE DI BYTE
0250: BD 00 03      LDA TABEG,X      ; CARICA LA NOTA DA ESEGUIRE
0253: B0 04          BCS ENDBYT      ; NIBBLE BASSO USATO, ACCETTA ALTO
0255: 29 0F          AND # %00001111; MASCHERA I BIT ALTI
0257: 90 06          BCC FINISH      ; ESEGUI LA NOTA
0259: 29 F0      ENDBYT AND # %11110000; NIBBLE BASSO
025B: 4A          LSR A              ; SCORRIMENTO IN BASSO
025C: 4A          LSR A
025D: 4A          LSR A
025E: 4A          LSR A
025F: 20 70 02 FINISH JSR PLAYIT      ; CALCOLA LE COSTANTI ED ESEGUI
0262: A2 20          LDX # $ 20      ; RITARDO TRA LE NOTE
0264: 20 9C 02      JSR DELAY
0267: E6 02          INC PTR          ; USATO UN NIBBLE
0269: A5 02          LDA PTR
026B: C5 00          CMP PILEN      ; FINE DELLA LISTA?
026D: 90 DF          BCC LOOP      ; NO, ACCETTA LA NOTA SUCCESSIVA
026F: 60          RTS              ; FATTO

; ROUTINE PER CONSULTAZIONE DI TABELLA, PAUSA SEPARATA

0270: C9 0D      PLAYIT CMP # 13      ; PAUSA?
0272: D0 06      BNE SOUND            ; NO.
0274: A2 54      LDX # $ 54          ; RITARDO = LUNGHEZZA NOTA = 0,21 SEC
0276: 20 9C 02      JSR DELAY
0279: 60          RTS
027A: AA          SOUND TAX          ; USA # TASTO COME INDICE ...
027B: BD D1 02      LDA DURTAB,X      ; ... PER TROVARE LA DURATA.
027E: 85 04          STA DUR          ; MEMORIZZA LA DURATA PER L'IMPIEGO
0280: BD C4 02      LDA NOTAB,X      ; CARICA IL VALORE DELLA NOTA
0283: 20 A8 02      JSR TONE
0286: 60          RTS

; ROUTINE PER ESEGUIRE UN SEGNALE DI TRE TONI

```

Fig. 2.9: Programma per la generazione della musica (continua)

```

0287: A9 FF      BEEP3 LDA # $ FF      ; DURATA DEI BEEP
0289: 85 04      STA DUR
028B: A9 4B      LDA # $ 4B      ; CODICE DI E2
028D: 20 A8 02   JSR TONE      ; PRIMA NOTA
0290: A9 38      LDA # $ 38      ; CODICE DI D2
0292: 20 A8 02   JSR TONE
0295: A9 4B      LDA # $ 4B
0297: 20 A8 02   JSR TONE
029A: 18        CLC
029B: 60        RTS

; RITARDO DI LUNGHEZZA VARIABILE

029C: A0 FF      DELAY LDY # $ FF
029E: EA        DLY NOP
029F: D0 00      BNE ,+ 2
02A1: 88        DEY
02A2: D0 FA      BNE DLY      ; CICLO DI 10 MICROSEC
02A4: CA        DEX
02A5: D0 F5      BNE DELAY    ; TEMPO DI CICLO = 2556 * [X]
02A7: 60        RTS

; ROUTINE PER ESEGUIRE UN TONO: IL # DI MEZZI CICLI
; È IN "DUR" ED IL TEMPO DI 1/2 CICLO È IN A.
; TEMPO DI CICLO = 20 * [A] + 26 MICROSEC
; POICHÉ DUE PASSAGGI DEL CICLO ESTERNO GENERANO
; UN CICLO DEL TONO

02A8: 85 03      TONE STA FREQ      ; LA FREQ È TEMP PER NUM DI CICLI
02AA: A9 FF      LDA # $ FF      ; PREDISPONI IL REG DI DIREZIONE DATI
02AC: 8D 02 AC   STA DDRB
02AF: A9 00      LDA # $ 00      ; A È INVIATO ALLA PORTA, START HI
02B1: A6 04      LD DUR
02B3: A4 03      FL2 LDY FREQ
02B5: 88        FL1 DEY
02B6: 18        CLC
02B7: 90 00      BCC ,+2
02B9: D0 FA      BNE FL1      ; CICLO INTERNO DI 10 MICROSEC
02BB: 49 FF      EOR # $ FF      ; COMPLEMENTA LA PORTA DI I/O
02BD: 8D 00 AC   STA OPB      ; ... E PREDISPONILA
02C0: CA        DEX
02C1: D0 F0      BNE FL2      ; CICLO ESTERNO
02C3: 60        RTS

; TABELLA DELLE COSTANTI DELLE NOTE
; CONTIENE:
; [OTTAVE SOTTO IL CENTRO C]: G,A,B
; [OTTAVE DEL CENTRO C]: C,D,E,F,F,#,G,G,#,A,B
; [OTTAVE SOPRA IL CENTRO C]: C
;
02C4: FE      NOTAB .BYT $FE,$E2,$C9,$BE,$A9,$96,$8E
02C5: E2
02C6: C9
02C7: 8E
02C8: A9
02C9: 96
02CA: 8E
02CB: 86      .BYT $86,$7E,$77,$70,$64,$5E
02CC: 7E
02CD: 77
02CE: 70
02CF: 64
02D0: 5E

; TABELLA DELLE DURATE DELLE NOTE IN # DI 1/2 CICLI
; PREDISPOSTA PER UNA LUNGHEZZA DELLA NOTA DI CIRCA 0,21 SEC.

```

Fig. 2.9: Programma per la generazione della musica (continua)

```

02D1: 55          DURTAB: BYT $55,$60,$6B,$72,$80,$8F,$94
02D2: 60
02D3: 6B
02D4: 72
02D5: 80
02D6: 8F
02D7: 94
02D8: A1          .BYT $A1,$AA,$B5,$BF,$D7,$E4
02D9: AA
02DA: B5
02DB: BF
02DC: D7
02DD: E4

```

SYMBOL TABLE:

GETKEY	0100	PILEN	0000	TEMP	0001
PTR	0002	FREQ	0003	DUR	0004
TABEG	0300	OPB	AC000	DDRB	AC02
START	0200	NXKEY	0205	NXTST	0211
NUMKEY	021B	OK	022B	FINBYT	023A
PLAYEM	0248	LOOP	024E	ENDBYT	0259
FINISH	025F	PLAYIT	0270	SOUND	027A
BEEP3	0287	DELAY	029C	DLY	029E
TONE	02A8	FL2	02B3	FL1	02B5
NOTAB	02C4	DURTAB	02D1		

%

Fig. 2.9: Programma per la generazione della musica (continua)

Ogni volta che viene premuto il tasto numero 15 viene eseguita una speciale routine di tre toni chiamata BEEP3. La routine BEEP3 si trova all'indirizzo 0287. Essa esegue tre note in rapida successione per indicare all'utente che le note che si trovavano in memoria sono state cancellate. La cancellazione viene eseguita dal reset a zero della lunghezza della lista PILEN. La routine corrispondente è la seguente:

BEEP3	LDA # \$FF	Costante di durata del beep
	STA DUR	
	LDA # \$4B	Codice di E2
	JSR TONE	Prima nota
	LDA # \$38	Codice di D2
	JSR TONE	Seconda nota
	LDA # \$4B	Codice di E2
	JSR TONE	Terza nota
	CLC	
	RTS	

Il suo modo di operare è immediato.

La routine NUMKEY salverà nella memoria il codice corrispondente alla nota. Come nel caso del programma della telescrivente, il computer invierà l'eco del carattere che è stato premuto sotto forma di un suono udibile. In altre parole, ogni volta che viene premuto un tasto, il programma esegue la nota corrispondente. Questo viene eseguito dalle due istruzioni successive:

```
NUMKEY    STA TEMP
          JSR PLAYIT
```

Si esegue quindi il controllo di overflow della lunghezza della lista. Se si verifica una situazione di overflow, si avverte l'operatore con la sequenza di tre toni BEEP3:

LDA PILEN	Accetta la lunghezza della lista
CMP #\$FF	Overflow?
BNE OK	No: aggiungi la nota alla lista
JSR BEEP3	Si: avverti l'operatore
BCC NXKEY	Leggi il tasto successivo

In caso contrario, il nuovo nibble (4 bit), corrispondente al numero di identificazione della nota, viene fatto scorrere nella lista:

OK	LSR A	Fa scorrere il bit basso nel puntatore del nibble
	TAY	Usalo come indice del byte
	LDA TEMP	Ripristina il numero del tasto

Notate che il puntatore del nibble viene diviso per due e diventa un indice del byte. Esso viene quindi memorizzato nel registro Y, che verrà impiegato successivamente per eseguire un accesso indicizzato alla locazione corretta del byte all'interno della tabella (STA TABEG,Y).

A seconda del valore del bit che è stato fatto scorrere nel carry, il nibble viene memorizzato nella parte alta o nella parte bassa dell'ingresso della tabella. Ogni volta che il nibble deve essere salvato nella posizione di ordine elevato del byte, è necessario uno scorrimento a sinistra di 4 bit, che richiede quattro istruzioni:

BCS	FINBYT	Test se il bit ha un nibble
AND	#\$00001111	Maschera il nibble superiore
STA	TABEG,Y	Salva
INC	PILEN	Nibble successivo
BCC	NXKEY	

```

FINBYT    ASL A
           ASL A
           ASL A
           ASL A

```

Infine è possibile salvarlo all'appropriato indirizzo della tabella:

```

ORA TABEG,Y
STA TABEG,Y

```

Il puntatore viene incrementato e viene esaminato il tasto successivo:

```

INC PILEN
BCC NXKEY

```

Esaminiamo questa tecnica con un esempio. Supponiamo:

```

PILEN = 9      (lunghezza della lista)
TEMP = 6       (tasto premuto)

```

L'effetto delle istruzioni è il seguente:

OK	LSR A	A conterrà 4, C conterrà 1
	TAY	Y = 4
	LDA TEMP	A = 6
	BCS FINBYT	C è 1 e si verifica la ramificazione

La situazione nella lista è:

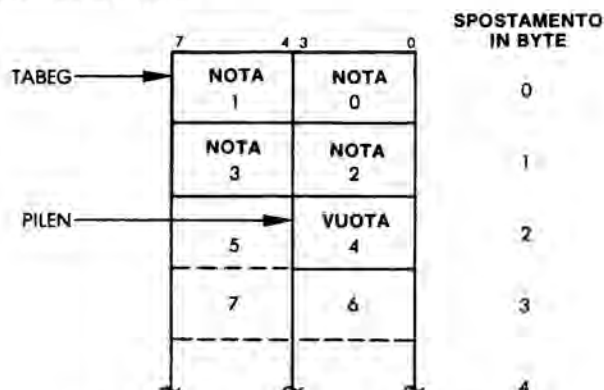


Figura 2.10: Ingresso di una nota nella lista

Si fa scorrere "6" nella posizione di ordine elevato di A:

```
FINBYT    ASL A
           ASL A
           ASL A
           ASL A           A = 60 (esadecimale)
```

Si scrive A nella tabella:

```
ORG TABEG, Y  A = 16X (dove X è il nibble
               precedente nella tabella)
```

```
STA TABEG,Y  Sostituisci il vecchio nibble
              con il nuovo
```

### Le subroutine

#### *Subroutine PLAYEM*

Anche la routine *PLAYEM* è immediata. La locazione di memoria PTR viene impiegata come puntatore corrente al nibble della tabella delle note. Come prima, i contenuti del puntatore corrente al nibble sono fatti scorrere a destra e diventano un puntatore di byte. Il corrispondente ingresso alla tabella viene quindi caricato con l'impiego di un metodo di indirizzamento indicizzato:

```
PLAYEM    LDX #0
           STX PTR           PTR=0
           LDA PTR
LOOP      LSR A
           TAX
           LDA TABEG,X
           BCS ENDBYT
           AND #%00001111
           BCC FINISH
ENDBYT    AND #%11110000
           LSR A
           LSR A
           LSR A
           LSR A
```

In dipendenza del valore del bit fatto scorrere nel carry, viene estratto il nibble di ordine elevato o di basso ordine e caricato a sinistra nell'accu-

mulatore. Per ottenere le costanti opportune e per eseguire la nota viene utilizzata la subrou tine PLAYIT descritta di seguito:

FINISH	JSR PLAYIT	Esegui la nota
--------	------------	----------------

Viene quindi implementato un ritardo tra due note consecutive, viene incrementato il puntatore corrente, si esegue un controllo di fine lista e si riesegue il ciclo:

LDX #\$20	Costante di ritardo
JSR DELAY	Ritardo tra le note
INC PTR	Impiegato un nibble
LDA PTR	
CMP PILEN	Controllo per fine lista
BCC LOOP	No: accetta un'altra nota
RTS	Fatto

### *Subroutine PLAYIT*

La subroutine PLAYIT esegue la nota oppure una pausa, secondo quanto specificato dal nibble che riceve dall'accumulatore. Questa subroutine   chiamata "PLAYNOTE" sul diagramma di flusso del programma. Essa preleva semplicemente la durata corretta della nota dalla tabella DURTAB e la salva all'indirizzo DUR (alla locazione di memoria 4). Successivamente carica il valore corretto del semi-periodo dalla tabella all'indirizzo NOTAB nel registro A, impiegando l'indirizzamento indicizzato, quindi chiama la subroutine TONE per eseguirlo:

PLAYIT	CMP #13	Controlla se c�
	BNE SOUND	No
	LDX #\$54	Ritardo=0,21 sec (durata nota)
	JSR DELAY	Se era specificata pausa
	RTS	
SOUND	TAX	Usa numero tasto come indice
	LDA DURTAB, X	Per ottenere la durata
	STA DUR	
	LDA NOTAB, X	
	JSR TONE	
	RTS	



## Subroutine TONE

La subroutine TONE implementa la procedura di generazione della forma d'onda corretta precedentemente descritta, ed attiva l'altoparlante alla frequenza corretta per eseguire la nota specifica. Essa implementa un ciclo di ritardo convenzionale a due livelli annidati ed attiva l'altoparlante complementando la porta d'uscita dopo che è trascorso ogni ritardo specifico:

TONE            STA FREQ

L'ingresso A contiene il valore del semi-periodo. Quindi viene memorizzato in FREQ. La temporizzazione del ciclo è tale da generare all'uscita una lunghezza d'onda pari a:

$$(20 \times A + 26) \mu s$$

La porta B viene configurata come uscita:

LDA #\$FF  
STA DDRB

Vengono quindi inizializzati i registri. A viene settato per contenere il pattern (configurazione) dell'uscita. X è il contatore del ciclo esterno. Viene posto uguale al valore DUR che contiene il numero di semi-periodi all'istante di chiamata della subroutine:

LDA #\$00  
LDX DUR

Viene quindi inizializzato il contatore del ciclo interno Y a FREQ, la costante di frequenza:

FL2            LDY FREQ

e viene generato normalmente il ritardo del ciclo interno:

FL1            DEY  
              CLC  
              BCC .+2  
              BNE FL1            Ciclo interno di 10  $\mu s$

Quindi viene fatta scattare la porta d'uscita mediante operazione di complemento:

```
    EOR #$FF  
    STA OPB
```

e viene completato il ciclo esterno:

```
    DEX  
    BNE FL2  
    RTS
```

La subroutine DELAY è riportata in Fig. 2.9, inizia alla locazione di memoria 29C ed il suo esame è lasciato come esercizio.

## SOMMARIO

Questo programma impiega un algoritmo molto semplice per la memorizzazione e l'esecuzione di accordi musicali. Tutti i dati e le costanti sono memorizzati sotto forma di tabelle. La temporizzazione viene implementata con cicli annidati. Per memorizzare e recuperare i dati vengono impiegate le tecniche di indirizzamento indicizzato. Il suono viene generato con un'onda quadra.

## ESERCIZI

**Esercizio 2-4:** *Cambiate le costanti delle note per implementare una diversa gamma di note.*

**Esercizio 2-5:** *Caricate in memoria un accordo in progressione. Attivatelo premendo il tasto "0".*

**Esercizio 2-6:** *Riscrivete il programma in modo che memorizzi le note e le costanti della durata in memoria quando vengono caricate, e non sia necessario prelevarle quando viene eseguito l'accordo. Quali sono gli svantaggi di questo metodo?*

## CAPITOLO 3

# TRADUCI

### LE REGOLE

Questo gioco viene condotto da due correnti. Ogni giocatore tenta di decifrare il più velocemente possibile i numeri codificati del computer. I giocatori tentano di indovinare a turno. Ogni volta il giocatore tenta di premere il tasto esadecimale corrispondente al numero di 4 bit visualizzato dal programma. Il programma tiene conto del tempo totale che un giocatore impiega per indovinare il numero. Il limite massimo è fissato in 17 secondi. Quando entrambi i giocatori hanno decodificato correttamente il numero, vengono confrontati i tempi di risposta per determinare chi vince il turno. Il giocatore che vince per primo dieci turni vince il gioco.

Il programma segnala il turno di ciascun giocatore visualizzando una freccia che punta a sinistra o a destra. Il gioco inizierà dal giocatore di destra. La Fig. 3.1 mostra il "prompt" del programma.

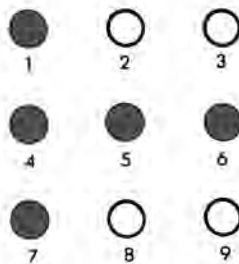


Figura 3.1: Segnale di pronto per il giocatore di destra

Dopo questo prompt del programma, trascorrerà un periodo di tempo casuale, quindi si illumina la riga in basso dei LED della Scheda Giochi. Il LED più a sinistra (LED #10) segnala al giocatore di precede-

re. I quattro LED più a destra (i LED 12, 13, 14 e 15) visualizzano il numero binario codificato. Questo è mostrato in Fig. 3.2. In questo caso, il giocatore 1 dovrebbe chiaramente premere il tasto 5. Se il giocatore 1 indovina, il programma passa al giocatore 2. Altrimenti il giocatore 1 avrà un'altra possibilità fino ad un massimo di 17 secondi complessivi. Si può osservare che, per ogni numero presentato al giocatore, viene accumulato un tempo massimo di circa 17 secondi. Quando si raggiunge il massimo, la riga in basso si spegne e viene visualizzato un nuovo numero.

Il programma segnala il secondo turno del giocatore (il giocatore a sinistra) visualizzando una freccia a sinistra sui LED come mostrato in Fig. 3.3. Una volta che entrambi i giocatori hanno avuto a disposizione il loro turno per indovinare un digit binario, il programma segnerà il vincitore illuminando i tre LED di destra o di sinistra della riga in basso. Il vincitore è il giocatore che ha impiegato il minor tempo. Il gioco si ripete fino a che un giocatore vincendo dieci volte ha vinto l'incontro. Il computer segnala il vincitore dell'incontro illuminando dieci volte i tre LED dal lato del corrispondente giocatore. Quindi il controllo viene restituito al SYM-1 monitor.

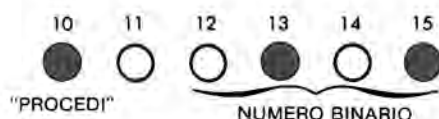


Figura 3.2: La riga in basso di LED visualizza il numero da indovinare

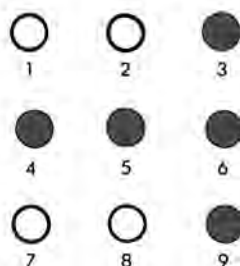


Figura 3.3: È il turno del giocatore 2 (giocatore di sinistra)

## UN GIOCO TIPICO

Viene visualizzata una freccia a destra. Appare una particolare disposizione dei LED sulla riga in basso: 10, 13, 14 e 15. Il giocatore a destra

(giocatore 1) preme il tasto "C", e si spegne la riga in basso di LED, giacchè la risposta non è esatta. Poichè il giocatore 1 non ha indovinato, ha la possibilità di effettuare un altro tentativo. Si illuminano i LED 13, 14 e 15 ed il giocatore preme il tasto "7". Questa volta la risposta è esatta e si illumina la freccia a destra, indicando il turno del secondo giocatore (giocatore 2). Questa volta il numero proposto è 10, 12 e 15. Il giocatore di sinistra preme il tasto "9". A questo punto si illuminano i LED 10, 11 e 12 indicando che il giocatore 2 è il vincitore del turno in quanto ha impiegato il minor tempo totale per fornire una risposta esatta.

Altro tentativo. Si illumina la freccia a destra: il numero da tradurre appare nei LED 10, 13, 14 e 15. Il giocatore 1 preme il tasto "7" ed appare una freccia a sinistra. Il numero successivo accende i LED 10 e 14. Il giocatore 2 preme il tasto "2". Anche questa volta si accendono i tre LED di sinistra in quanto il giocatore 2 è stato più veloce del giocatore 1 nel fornire la risposta corretta.

## L'ALGORITMO

La Fig. 3.4 riporta il diagramma di flusso corrispondente al programma. Viene implementato un primo ciclo di attesa per misurare il tempo impiegato dal giocatore 1 per rispondere correttamente. Una volta che il giocatore 1 ha risposto correttamente, il suo tempo totale viene accumulato in una variabile chiamata TEMP. È quindi la volta del giocatore 2, viene implementato un ciclo di attesa analogo. Quando entrambi i giocatori hanno indovinato vengono confrontati i tempi rispettivi. Vince il giocatore che ha impiegato il minor tempo con l'indicazione a sinistra o a destra, come indicato dalle label 1 e 2 sul diagramma di flusso di Fig. 3.4. Per il conteggio del numero di giochi vinti da un giocatore specifico viene impiegata una variabile secondaria PLYR1 oppure PLYR2. Ad ogni vincita di un giocatore questa variabile viene incrementata e si verifica se si è raggiunto il valore 10. In caso contrario si inizia un nuovo gioco. Se si raggiunge il valore 10, il giocatore con questo punteggio viene dichiarato vincitore dell'incontro.

## IL PROGRAMMA

Il programma corrispondente impiega soltanto una struttura dati significativa. Si chiama NUMTAB e viene impiegata per facilitare la visualizzazione del numero binario casuale sui LED. Si ricordi che il LED # 10 deve essere sempre illuminato (è il LED che indica di procedere). Invece il LED 11 deve essere sempre spento. I LED 12, 13, 14 e 15 vengono impiegati per visualizzare il numero binario. Si ricordi inoltre che la posizione di bit 6 della Porta 1B non viene utilizzata. Conseguente-

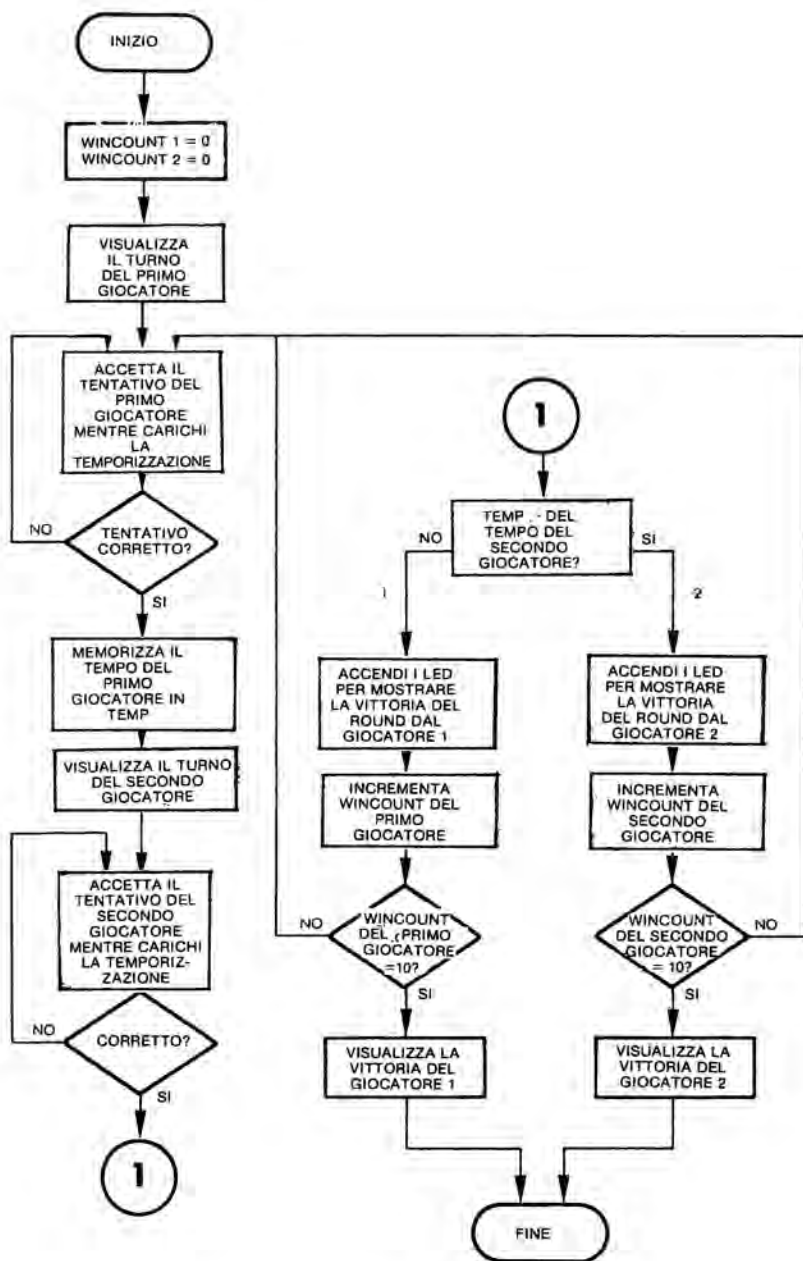
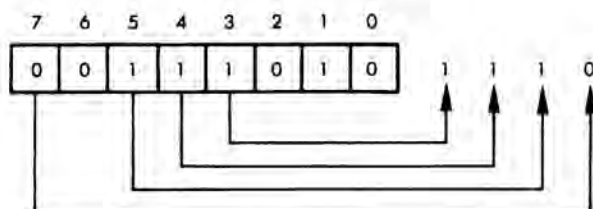


Figura 3.4: Diagramma di flusso di Traduci

mente la visualizzazione di uno "0" si ottiene inviando in uscita il pattern "00000010". Invece la visualizzazione di un "1" viene ottenuta con la configurazione "10000010". La visualizzazione di un "3" si ottiene con "10100010", e così via. (Vedere Fig. 3.5).

Le configurazioni complete corrispondenti a tutte le sedici possibilità sono memorizzate nella tabella NUMTAB (Vedere la Fig. 3.6). Esaminiamo, per esempio, l'ingresso 14 di NUMTAB (Vedere la linea 0060 del programma). Esso vale "00111010". Il numero binario corrispondente da visualizzare sarà perciò: "00111".



Si tratta di "1110" ovvero 14. Si ricordi che il bit 6 di questa porta è sempre "0".

### Area di memoria inferiore

Le locazioni di memoria da 0 a 10 vengono impiegate per memorizzare le variabili temporanee e la tabella NUMTAB. Le funzioni delle variabili sono:

TEMP	Memorizzazione per la lunghezza casuale del ritardo
CNTHI, CNTLO	Tempo impiegato da un giocatore per premere il tasto
CNTIH, CNTIL	Tempo impiegato dal giocatore 1 per premere il tasto (memorizzazione permanente)
PLYR1	Punteggio del giocatore 1 (numero di turni vinti fino a quel momento, fino ad un massimo di dieci)
PLYR2	Stessa cosa per il giocatore 2
NUMBER	Numero casuale da indovinare
SCR e successive	Area Scratch utilizzata dal generatore di numeri casuali

VIA #1

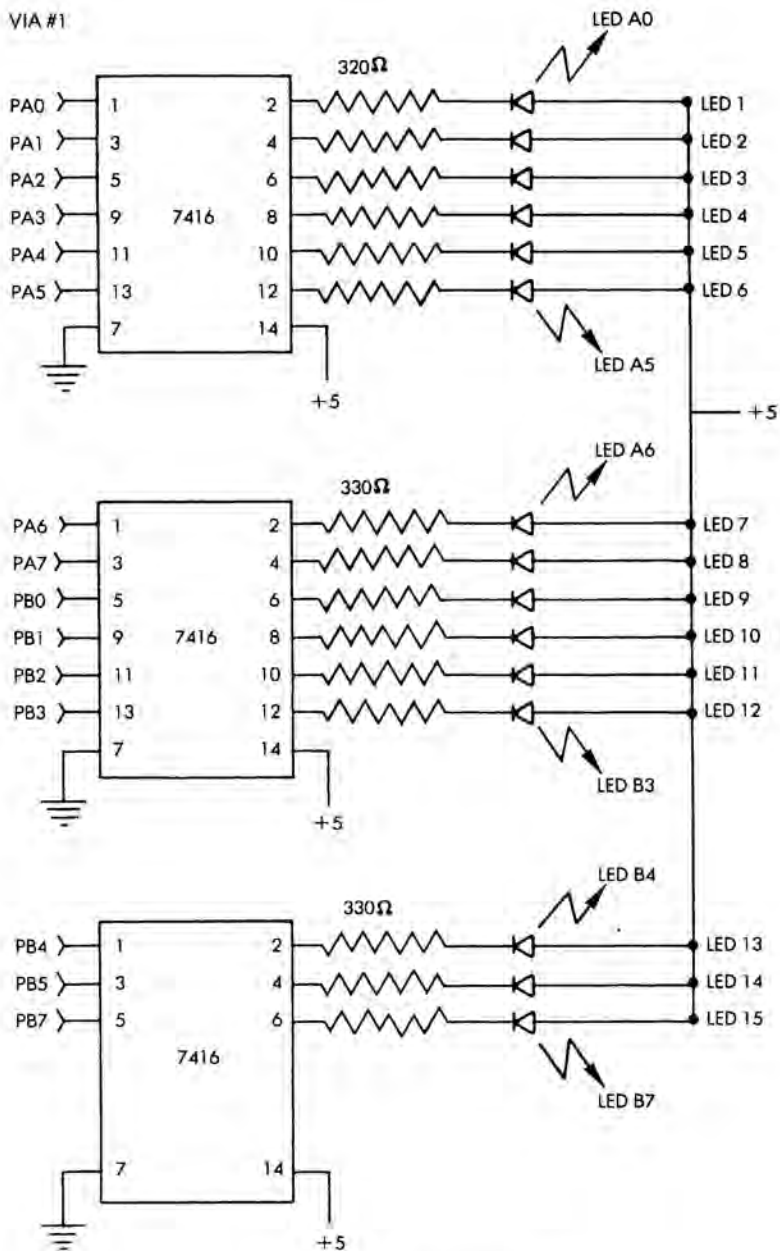


Figura 3.5: Connessioni dei LED



Nell'assembler listing, il metodo impiegato per riservare le locazioni di memoria, in questo programma, è diverso dal metodo impiegato dal programma del Capitolo 2. Nel programma del GIOCO MUSICALE la memoria è stata riservata alle variabili dichiarando semplicemente il valore dei simboli che rappresentano le locazioni delle variabili, per mezzo dell'istruzione:

<NOME DELLA VARIABILE> = <INDIRIZZO DI MEMORIA>

In questo programma il contatore delle locazioni dell'assemblatore viene incrementato mediante espressioni del tipo:

$$* = * + n$$

Sostanzialmente, i simboli delle locazioni delle variabili in questo programma sono dichiarati come "label", mentre nel programma MUSICALE sono dei "simboli" veri e propri o "simboli di costanti".

Il programma di questo capitolo è costituito da una routine principale, chiamata MOVE e da cinque subroutine: PLAY, COUNTER, BLINK, DELAY e RANDOM. Esaminiamole. I registri di direzione dati A e B dei VIA # 1 e # 3 della scheda devono essere preliminarmente inizializzati. DDR1A, DDR1B e DDR3B sono configurati come uscite:

```
START      LDA #$FF
            STA DDR1A
            STA DDR1B
            STA DDR3B
```

DDR3A è condizionato come ingresso:

```
LDA #0
STA DDR3A
```

Infine vengono inizializzate a zero le variabili PLYR1 e PLYR2, utilizzate per accumulare il numero di vincite di ciascun giocatore:

```
STA PLYR1
STA PLYR2
```

Si entra quindi nel corpo principale di MOVE. Verrà visualizzata una freccia a destra per indicare il turno del giocatore 2. La Fig. 3.5 mostra le connessioni dei LED. Per visualizzare una freccia a destra, occorre illuminare i LED 1, 4, 5, 6 e 7 (si faccia riferimento anche alla Fig. 3.1). Questo viene ottenuto inviando il codice appropriato alla Porta 1A:

MOVE

LDA #%01111001

STA PORT1A

Visualizza una freccia a destra

La linea in basso di LED deve essere spenta:

LDA #0

STA PORT1B

Infine, occorre azzerare i contatori che misurano il tempo trascorso:

STA CNTLO

STA CNTHI

Siano pronti per iniziare il gioco:

JSR PLAY

La routine PLAY verrà descritta più avanti. Essa ritorna alla routine chiamante con la misura del tempo trascorso immagazzinata nelle locazioni CNTLO e CNTHI.

Supponiamo di tornare al programma principale (linea 0082 della Fig. 3.6). La durata del tempo trascorso che è stata accumulata nelle locazioni CNTLO e CNTHI dalla routine PLAY viene salvata in un set di locazioni permanenti riservate al giocatore 1 e chiamate CNT1L, CNT1H:

LDA CNTLO

STA CNT1L

LDA CNTHI

STA CNT1H

Successivamente è il turno del giocatore 2 e viene visualizzata una freccia a sinistra. Questo viene ottenuto accendendo i LED 3, 4, 5 e 6:

LDA #%000111100 Visualizza una freccia a  
sinistra

STA PORT1A

Quindi viene acceso il LED #9 per completare la freccia a sinistra:

LDA #1

STA PORT1B

LINE #	LOC	CODE	LINE
0002	0000		: 'TRADUCI'
0003	0000		: PROGRAMMA PER IL TEST DELLA VELOCITA' DI 2 GIOCATORI
0004	0000		: NELLA TRADUZIONE DI UN NUMERO BINARIO IN UN SINGOLO
0005	0000		: DIGIT ESADECIMALE. IL TURNO DI CIASCUN GIOCATORE
			: VIENE
0006	0000		: MOSTRATO DA UN PUNTATORE A SINISTRA
0007	0000		: O A DESTRA.
0008	0000		: IL NUMERO VERRA' RAPIDAMENTE LAMPEGGIATO
0009	0000		: SUI LED 12-15, ACCOMPAGNATO DALL'ACCENSIONE DEL
			: LED # 10.
0010	0000		: IL GIOCATORE DEVE QUINDI PREMERE IL PULSANTE
			: CORRISPONDENTE.
0011	0000		: DOPO I TENTATIVI DI ENTRAMBI I GIOCATORI, I RISULTATI
0012	0000		: VENGONO MOSTRATI SULLA RIGA IN BASSO.
0013	0000		: DOPO 10 VITTORIE, VENGONO LAMPEGGIATI I RISULTATI
0014	0000		: DI UN GIOCATORE, INDICANDO IL MIGLIOR GIOCATORE.
0015	0000		: QUINDI IL GIOCO RICOMINCIA
0016	0000		
0017	0000		: I/O:
0018	0000		
0019	0000		PORT1A = \$A001 ; LED 1-8
0020	0000		PORT1B = \$A000 ; LED 9-15
0021	0000		DDR1A = \$A003
0022	0000		DDR1B = \$A002
0023	0000		PORT3A = \$AC01 ; INGRESSO STROBE DEL TASTO
0024	0000		PORT3B = \$AC00 ; USCITA # TASTO
0025	0000		DDR3A = \$AC03
0026	0000		DDR3B = \$AC02
0027	0000		
0028	0000		: MEMORIZZAZIONE DELLE VARIABILI:
0029	0000		
0030	0000		*+\$0
0031	0000		
0032	0000		TEMP *="+1
0033	0001		CNTHI *="+1 ; MEMORIZZAZIONE TEMPORANEA DI
			: AMT. DEL
0034	0002		: TEMPO IMPIEGATO DAL GIOCATORE PER
			: INDOVINARE
0035	0002		CNTL0 *="+1
0036	0003		CNT1H *="+1 ; AMT. DEL TEMPO IMPIEGATO DAL
			: GIOCATORE 1 PER INDOVINARE
0037	0004		CNT1L *="+1
0038	0005		PLYR1 *="+1 ; PUNTEGGIO O NUMERO DI VITTORIE
			: PER PLYR1
0039	0006		PLYR2 *="+1 ; PUNTEGGIO DEL GIOCATORE 2
0040	0007		NUMBER *="+1 ; MEMORIZZA NUMERO DA
			: INDOVINARE
0041	0008		SCR *="+6 ; SCRATCHPAD PER GENERATORE
			: NUM. CASUALI
0042	000E		
0043	000E		: TABELLA DEI NUMERI 'INVERSI' PER IL DISPLAY
0044	000E		: NEI BIT 3-8 DELLA PORTA1B, OPPURE LED 12-15
0045	000E		
0046	000E	02	NUMTAB.BYTE %00000010
0047	000F	82	.BYTE %10000010
0048	0010	22	.BYTE %00100010
0049	0011	A2	.BYTE %10100010
0050	0012	12	.BYTE %00010010
0051	0013	92	.BYTE %10010010
0052	0014	32	.BYTE %00110010
0053	0015	B2	.BYTE %10110010

Figura 3.6: Programma Traduci

```

0054 0016 0A .BYTE %00001010
0055 0017 8A .BYTE %10001010
0056 0018 2A .BYTE %00101010
0057 0019 AA .BYTE %10101010
0058 001A 1A .BYTE %00011010
0059 001B 9A .BYTE %10011010
0060 001C 3A .BYTE %00111010
0061 001D BA .BYTE %10111010
0062 001E
0063 001E ; PROGRAMMA PRINCIPALE
0064 001E
0065 001E *= $200
0066 0200
0067 0200 A9 FF START LDA # $ FF ; IMPOSTA LE PORTE
0068 0202 8D 03 A0 STA DDR1A
0069 0205 8D 02 A0 STA DDR1B
0070 0208 8D 02 AC STA DDR3B
0071 020B A9 00 LDA # 0
0072 020D 8D 03 AC STA DDR3A
0073 0210 85 05 STA PLYR1 ; AZZERA NUMERO DI VITTORIE
0074 0212 85 06 STA PLYR2
0075 0214 A9 79 MOVE LDA # %01111001
0076 0216 8D 01 A0 STA PORT1A ; VISUALIZZA FRECCIA A DESTRA.
0077 0219 A9 00 LDA # 0
0078 021B 8D 00 A0 STA PORT1B
0079 021E 85 02 STA CNTLO ; AZZERA CONTATORI.
0080 0220 85 01 STA CNTHI
0081 0222 20 8C 02 JSR PLAY ; ACCETTA IL TEMPO DEL PRIMO
; GIOCATORE.
0082 0225 A5 02 LDA CNTLO ; TRASFERISCI IL CONTEGGIO
; TEMPO ALLA MEM. PERMANENTE
0083 0227 85 04 STA CNT1L
0084 0229 A5 01 LDA CNTHI
0085 022B 85 03 STA CNT1H
0086 022D A9 3C LDA # $00111100 ; VISUALIZZA FRECCIA A SINISTRA.
0087 022F 8D 01 A0 STA PORT1A
0088 0232 A9 01 LDA # 1
0089 0234 8D 00 A0 STA PORT1B
0090 0237 A9 00 LDA # 0
0091 0239 85 02 STA CNTLO ; AZZERA CONTATORI.
0092 023B 85 01 STA CNTHI
0093 023D 20 8C 02 JSR PLAY ; ACCETTA TEMPO GIOCATORE 2.
0094 0240 A5 01 LDA CNTHI ; ACCETTA CONTEGGIO
; GIOCATORE 2 E ...
0095 0242 C5 03 CMP CNT1H ; CONFRONTALO CON QUELLO
; DEL PRIMO GIOCATORE
0096 0244 F0 04 BEQ EQUAL ; CONTROLLA I BYTE DI BASSO
; ORDINE PER DET. VINCITORE.
0097 0246 90 27 BCC PLR2 ; GIOCATORE 2 HA PUNTEGGIO
; PIU' PICCOLO, VISUALIZZALO.
0098 0248 B0 08 BCS PLR1 ; GIOCATORE 1 HA PUNTEGGIO
; PIU' PICCOLO, VISUALIZZALO.
0099 024A A5 02 EQUAL LDA CNTLO ; BYTE DI ORDINE ELEVATO SONO
; UGUALI. QUINDI
0100 024C ; CONTROLLA QUELLI BASSI.
0101 024C C5 04 CMP CNT1L ; CONFRONTA I PUNTEGGI.
0102 024E 90 1F BCC PLR2 ; VINCE IL GIOCATORE 2,
; VISUALIZZALO.
0103 0250 B0 00 BCS PLR1 ; VINCE IL GIOCATORE 1,
; VISUALIZZALO.
0104 0252 A9 F0 PLR1 LDA # %11110000 ; ACCENDI PARTE DESTRA RIGA IN
; BASSO

```

Figura 3.6: Programma Traduci (continua)

0105	0254	8D 00 A0	STA PORT1B	; PER MOSTRARE IL VINCITORE.
0106	0257	A9 00	LDA # 0	
0107	0259	8D 01 A0	STA PORT1A	; AZZERA I LED BASSI.
0108	025C	A9 40	LDA # \$ 40	; ATTENDI MENTRE VISUALIZZI IL VINCITORE.
0109	025E	20 E3 02	JSR DELAY	
0110	0261	E6 05	INC PLYR1	; IL GIOCATORE 1 VINCE UN'ALTRA VOLTA ...
0111	0263	A9 0A	LDA # 10	; ... HA VINTO 10 VOLTE?
0112	0265	C5 05	CMP PLYR1	
0113	0267	D0 AB	BNE MOVE	; SE NO, ESEGUI UN ALTRO ROUND.
0114	0269	A9 F0	LDA #%11110000	; SI, ACCETTA LA STRUTTURA DI LAMPEGGIO.
0115	026B	20 CB 02	JSR BLINK	; LAMPEGGIA IL LATO VINCENTE.
0116	026E	60	RTS	; FINE GIOCO: RITORNO AL MONITOR.
0117	026F	A9 0E PLR2	LDA #%1110	; ACCENDI IL LATO SINISTRO IN BASSO.
0118	0271	8D 00 A0	STA PORT1B	
0119	0274	A9 00	LDA # 0	
0120	0276	8D 01 A0	STA PORT1A	; AZZERA I LED BASSI.
0121	0279	A9 40	LDA # \$ 40	; ATTENDI MENTRE VISUALIZZI IL VINCITORE.
0122	027B	20 E3 02	JSR DELAY	
0123	027E	E6 06	INC PLYR2	; IL GIOCATORE 2 HA VINTO UN ALTRO ROUND ...
0124	0280	A9 0A	LDA # 10	; ... NE HA VINTI 10?
0125	0282	C5 06	CMP PLYR2	
0126	0284	D0 8E	BNE MOVE	; SE NO, ESEGUI UN ALTRO ROUND.
0127	0286	A9 0E	LDA #%1110	; SI, ACCETTA LA STRUTTURA PER LAMPEGGIARE I LED.
0128	0288	20 CB 02	JSR BLINK	; FALLI LAMPEGGIARE.
0129	028B	60	RTS	; FINE
0130	028C			
0131	028C			
0132	028C			
0133	028C			
0134	028C			
0135	028C			
0136	028C			
0137	028C	20 F4 02	PLAY JSR RANDOM	; ACCETTA IL NUMERO CASUALE.
0138	028F	20 E3 02	JSR DELAY	; RITARDO DI LUNGHEZZA CASUALE.
0139	0292	20 F4 02	JSR RANDOM	; ACCETTANE UN ALTRO.
0140	0295	29 0F	AND # \$ 0F	; MANTIENILO INFERIORE A 16 PER L'IMPIEGO COME
0141	0297	85 07	STA NUMBER	; NUMERO DA INDOVINARE.
0142	0299	AA	TAX	; USALO COME INDICE A ...
0143	029A	B5 0E	LDA NUMTAB,X	; ... ACCETTA LA STRUTTURA INVERSA DALLA TABELLA ...
0144	029C	0D 00 A0	ORA PORT1B	; PER VISUALIZZARE NEI LED 12-15.
0145	029F	8D 00 A0	STA PORT1B	
0146	02A2	20 B5 02	JSR CNTSUB	; ACCETTA LA PRESSIONE TASTO ED IL CONTEGGIO DURATA.
0147	02A5	C4 07	CPY NUMBER	; LA PRESSIONE TASTO È UN TENTATIVO CORRETTO?
0148	02A7	F0 0B	BEQ DONE	; SE SÌ, FATTO.
0149	02A9	A9 01	LDA # 01	; NO: CANCELLA IL VECCHIO TENTATIVO DAI LED.
0150	02AB	2D 00 A0	AND PORT1B	
0151	02AE	8D 00 A0	STA PORT1B	
0152	02B1	4C 8C 02	JMP PLAY	; PROVA ANCORA CON UN ALTRO NUMERO.

Figura 3.6: Programma Traduci (continua)

0153	02B4	60	DONE	RTS		; RITORNA CON LA DURATA IN
						; CNTLO + CNTHI
0154	02B5					
0155	02B5					; SUBROUTINE 'COUNTER'
0156	02B5					; ACCETTA LA PRESSIONE DEL TASTO MENTRE CONSERVA
						; TRACCIA
0157	02B5					; DELL'AMT DEL TEMPO PRIMA DELLA PRESSIONE TASTO.
0158	02B5					
0159	02B5	A0 0F	CNTSUB	LDY # \$ F		; PREDISPONI IL CONTATORE DEL
						; # TASTO.
0160	02B7	8C 00	AC	KEYLP	STY	PORT3B
						; USCITA DEL # TASTO AL MPXR
						; DELLA TASTIERA.
0161	02BA	2C 01	AC		BIT	PORT3A
0162	02BD	10 0B			BPL	FINISH
						; TASTO PREMUTO?
0163	02BF	88			DEY	
						; CONTEGGIO # TASTO PREMUTO.
0164	02C0	10 F5			BPL	KEYLP
						; PROVA UN ALTRO TASTO.
0165	02C2	E6 02			INC	CNTLO
						; VERIFICATI TUTTI I TASTI.
						; INCREMENTA CONTEGGIO.
0166	02C4	D0 EF			BNE	CNTSUB
						; SE NON OVERFLOW PROVA
						; ANCORA I TASTI.
0167	02C6	E6 01			INC	CNTHI
						; INCREMENTA IL BYTE LATO PER
						; OVERFLOW.
0168	02C8	D0 EB			BNE	CNTSUB
						; PROVA ANCORA I TASTI.
0169	02CA	60	FINISH	RTS		; FATTO: TEMPO SCADUTO OPPURE
						; TASTO PREMUTO.
0170	02CB					
0171	02CB					; SUBROUTINE 'BLINK'
0172	02CB					; ESEGUE IL LAMPEGGIO DEI LED I CUI BIT SONO POSTI
						; NELL'ACCUMULATORE
						; AL MOMENTO DELLA CHIAMATA.
0173	02CB					
0174	02CB					
0175	02CB	A2 14	BLINK	LDX # 20		; 20 LAMPEGGI.
0176	02CD	86 01		STX	CNTHI	; PREDISPONI IL CONTATORE DEI
						; LAMPEGGI.
0177	02CF	85 02		STA	CNTLO	; REGISTRO DI LAMPEGGIO.
0178	02D1	A5 02	BLOOP	LDA	CNTLO	; ACCETTA LA STRUTTURA DI
						; LAMPEGGIO.
						; LAMPEGGIA I LED.
0179	02D3	4D 00	A0	EOR	PORT1B	
0180	02D6	8D 00	A0	STA	PORT1B	
0181	02D9	A9 0A		LDA	# 10	; RITARDO BREVE.
0182	02DB	20 E3	02	JSR	DELAY	
0183	02DE	C6 01		DEC	CNTHI	
0184	02E0	D0 EF		BNE	BLOOP	; RICICLA SE NON FATTO.
0185	02E2	60		RTS		
0186	02E3					
0187	02E3					; SUBROUTINE 'DELAY'
0188	02E3					; I CONTENUTI DEL REG. A DETERMINANO LA LUNGHEZZA
						; DEL RITARDO.
0189	02E3					
0190	02E3	85 00	DELAY	STA	TEMP	
0191	02E5	A0 10	DL1	LDY	# \$ 10	
0192	02E7	A2 FF	DL2	LDX	# \$ FF	
0193	02E9	CA	DL3	DEX		
0194	02EA	D0 FD		BNE	DL3	
0195	02EC	88		DEY		
0196	02ED	D0 FB	BNE	DL2		
0197	02EF	C6 00		DEC	TEMP	
0198	02F1	D0 F2		BNE	DL1	
0199	02F3	60		RTS		
0200	02F4					
0201	02F4					; SUBROUTINE 'RANDOM'
0202	02F4					; GENERATORE DI NUMERI CASUALI.
0203	02F4					; RESTITUISCE IL NUMERO CASUALE NELL'ACCUMULATORE.

Figura 3.6: Programma Traduci (continua)

```

0204 02F4
0205 02F4 38 ; RANDOMSEC
0206 02F5 A5 09 LDA SCR+1
0207 02F7 65 0C ADC SCR+4
0208 02F9 65 0D ADC SCR+5
0209 02FB 85 08 STA SCR
0210 02FD A2 04 LDX # 4
0211 02FF B5 08 RNDLP LDA SCR,X
0212 0301 95 09 STA SCR+1,X
0213 0303 CA DEX
0214 0304 10 F9 BPL RNDLP
0215 0306 60 RTS
0216 0307 .END

```

#### SYMBOL TABLE

#### SYMBOL VALUE

BLINK	02CB	BLOOP	02D1	CNT1H	0003	CNT1L	0004
CNTHI	0001	CNTLO	0002	CNTSUB	02B5	DDR1A	A003
DDR1B	A002	DDR3A	AC03	DDR3B	AC02	DELAY	02E3
DL1	02E5	DL2	02E7	DL3	02E9	DONE	02B4
EQUAL	024A	FINISH	02CA	KEYLP	02B7	MOVE	0214
NUMBER	0007	NUMTAB	000E	PLAY	028C	PLR1	0252
PLR2	026F	PLYR1	0005	PLYR2	0006	PORT1A	A001
PORT1B	A000	PORT3A	AC01	PORT3B	AC00	RANDOM	02F4
RNDLP	02FF	SCR	0008	START	0200	TEMP	0000

END OF ASSEMBLY

Figura 3.6: Programma Traduci (continua)

Analogamente a prima avviene il reset a zero del contatore del tempo trascorso;

```

LDA #0
STA CNTLO
STA CNTHI

```

ed il giocatore 2 può giocare:

```
JSR PLAY
```

Quindi il tempo impiegato dal giocatore 2 viene confrontato a quello del giocatore 1. Se vince il giocatore 2, si verifica una ramificazione a PLR2. Se vince il giocatore 1 la ramificazione sarà PLR1. I byte di ordine elevato vengono confrontati per primi. Qualora essi siano uguali, vengono confrontati i byte di basso ordine:

```

LDA CNTHI
CMP CNTIH
BEQ EQUAL

```

Confronta i byte di

	BCC PLR2	Giocatore 2 ha tempo inferiore?
EQUAL	BCS PLR1	Oppure il giocatore 1?
	LDA CNTLO	Confronta i due byte
	CMP CNTIL	
	BCC PLR2	
	CMP CNTIL	
	BCC PLR2	
	BCS PLR1	

Una volta identificato il vincitore, viene illuminata la riga di LED che punta al vincitore. Vediamo cosa succede quando vince PLR1, per esempio. Vengono illuminati i tre LED all'estrema destra (i LED da 13 a 15):

```
PLR1      LDA #%11110000
          STA PORT1B
```

Gli altri LED sulla Scheda Giochi vengono spenti:

```
LDA #0
STA PORT1A
```

Viene quindi implementato un DELAY (ritardo) e siamo pronti per eseguire un altro gioco, fino ad un totale di dieci:

```
LDA #$40
JSR DELAY
```

Viene quindi incrementato il punteggio del giocatore 1:

```
INC PLYR1
```

Si confronta il punteggio del giocatore con 10. Se è inferiore a 10 si ha un ritorno alla routine principale MOVE:

```
LDA #10
CMP PLYR1
BNE MOVE
```

Altrimenti, è stato raggiunto il punteggio massimo di 10 ed il gioco è terminato. In questo caso vengono fatti lampeggiare i LED dal lato del vincitore:



```
LDA #%11110000    Pattern di lampeggio
JSR BLINK
RTS
```

La sequenza corrispondente per il giocatore 2 è riportata all'indirizzo PLR2 (linea 117 in Fig. 3.6):

```
PLR2      LDA #%1110
          STA PORT1B
          LDA #0
          STA PORT1A
          LDA #$40
          JSR DELAY
          INC PLYR2
          LDA #10
          CMP PLYR2
          BNE MOVE
          LDA #%1110
          JSR BLINK
          RTS
```

## Le subroutine

### *Subroutine PLAY*

La subroutine PLAY preliminarmente attende un periodo di tempo casuale prima di visualizzare il numero binario. Questo viene ottenuto chiamando la subroutine RANDOM per ottenere il numero casuale, quindi la subroutine DELAY implementa il ritardo:

```
PLAY      JSR RANDOM
          JSR DELAY
```

La subroutine RANDOM verrà descritta in seguito. Viene quindi ottenuto un altro numero casuale. Esso viene scelto tra 0 e 15 compresi. Questo sarà il numero binario visualizzato sui LED. Esso viene memorizzato alla locazione NUMBER:

```
JSR RANDOM
ABD #0F      Poni a zero il nibble elevato
STA NUMBER
```

La tabella NUMTAB, descritta all'inizio del capitolo 1, viene quindi impiegata per ottenere il pattern corretto di illuminazione dei LED con l'impiego dell'indirizzamento indicizzato. Il registro X contiene il numero tra 0 e 15 da visualizzare:

TAX	Usa X come indice
LDA NUMTAB,X	Recupera il pattern

Il pattern nell'accumulatore viene quindi memorizzato nel registro d'uscita in modo da illuminare i LED. Si noti che viene eseguito l'OR del pattern con i contenuti precedenti del registro d'uscita, cosicchè lo stato del LED 9 non viene cambiato:

```
ORA PORTIB
STA PORTIB
```

Una volta che il numero casuale è stato visualizzato in forma binaria sui LED, la subroutine attende finchè il giocatore ha premuto un tasto. Per questo scopo viene impiegata la subroutine CNTSUB:

```
JSR CNTSUB
```

Anche questa subroutine verrà descritta in seguito.

Il valore restituito nel registro Y da questa subroutine viene confrontato con il numero da indovinare, che è memorizzato all'indirizzo di memoria NUMBER. Se il confronto è positivo, si esce. In caso contrario tutti i LED vengono spenti con un'operazione di AND, in modo da prevenire variazioni dello stato del LED 9 e si rientra nella subroutine. Si noti che il tempo rimanente per il giocatore sarà decrementato ogni volta che viene chiamata la subroutine CNTSUB. Qualora il decremento raggiunga lo zero, verrà assegnato a questo giocatore un altro numero da indovinare:

	CPY NUMBER	Tentativo corretto?
	BEQ DONE	
	LDA #01	No: cancella il vecchio tentativo
	AND PORTIB	
	STA PORTIB	
	JMP PLAY	Prova ancora
DONE	RTS	

**Esercizio 3-1:** Si modifichi PLAY e/o CNTSUB in modo che, allo scadere del tempo, il giocatore perda la mano, come se fosse trascorso il tempo massimo per indovinare il numero.

### *Subroutine CNTSUB*

La subroutine CNTSUB viene impiegata dalla subroutine PLAY precedentemente descritta. Essa esegue il monitoraggio del tasto premuto dal giocatore e registra il tempo trascorso fino alla pressione del tasto. L'esplorazione del tasto viene eseguita nel modo solito:

CNTSUB	LDY #\$F	
KEYLP	STY PORT3B	
	BIT PORT3A	
	BPL FINISH	
	DEY	Conto alla rovescia numerato
		tasto
	BPL KEYLP	Tasto numerato
FINISH	BNE CNTSUB	

Ogni volta che tutti i tasti sono stati esplorati senza successo, viene incrementato il contatore del tempo trascorso (CNTLO, CNTHI):

	INC CNTLO
	BNE CNTSUB
	INC CNTHI
	BNE CNTSUB
FINISH	RTS

Al ritorno della subroutine, il numero corrispondente al tasto premuto è contenuto nel registro indice Y.

**Esercizio 3-2:** *Inserite alcune istruzioni "nessuna operazione" nella subroutine CNTSUB in modo da aumentare il tempo per il tentativo.*

### *Subroutine BLINK*

I LED specificati dai contenuti dell'accumulatore vengono fatti lampeggiare (accesi e spenti) dieci volte da questa subroutine. Essa usa la locazione di memoria CNTHI e CNTLO come registri scratch e distrugge i loro contenuti precedenti. Poichè i LED devono essere accesi e spenti alternativamente, viene impiegata un'istruzione di OR esclusivo per fornire l'accensione e spegnimento automatico, eseguendo un'operazione di complemento. Poichè, per eseguire il lampeggio dei LED una volta, occorrono due operazioni di complemento sullo stato dei LED, il ciclo viene eseguito 20 volte. Notate inoltre che i LED devono essere mantenuti accesi per un tempo minimo. Il programma è il seguente:

BLINK	LDX #20	20 lampeggi
	STX CNTHI	Contatore dei lampeggi
	STA CNTLO	Registro dei lampeggi
BLOOP	LDA CNTLO	Accetta la struttura di lampeggio
	EOR PORTIB	Lampeggia i LED
	STA PORTIB	
	LDA #10	Ritardo corto
	JSR DELAY	
	DEC CNTHI	
	BNE BLOOP	Ricicla se non fatto
	RTS	

### *Subroutine DELAY*

La subroutine DELAY implementa un classico progetto di ciclo annidato a tre livelli. Il registro X viene impostato al valore massimo FF (esadecimale) ed impiegato come contatore del ciclo interno. Il registro Y viene posto al valore 10 (esadecimale) ed utilizzato come contatore del ciclo a 2 livelli. La locazione TEMP contiene il numero impiegato per regolare il ritardo ed è il contatore del ciclo più esterno. La realizzazione della subroutine è immediata:

```

DELAY    STA TEMP
DL1      LDY #$10
DL2      LDX #$FF
DL3      DEX
          BNE DL3
          DEY
          BNE DL2
          DEC TEMP
          BNE DL1
          RTS

```

**Esercizio 3-3:** Calcolate la durata esatta del ritardo implementato da questa subroutine in funzione del numero contenuto nella locazione TEMP.

### *Subroutine RANDOM*

Questo semplice generatore di numeri casuali ritorna con un numero semi-casuale nell'accumulatore. Un set di sei locazioni a partire dall'indirizzo di memoria 0008 ("SCR") è stato riservato come scratch-pad

(memoria di lavoro) di questo generatore. Il numero casuale viene calcolato come 1 più il contenuto del numero nella locazione SCR + 1, più i contenuti del numero nella locazione SCR + 4, più il numero contenuto nella locazione SCR + 5:

```

RANDOM    SEC
          LDA SCR + 1
          ADC SCR + 4
          ADC SCR + 5
          STA SCR
    
```

I contenuti dell'area scratch (SCR e locazioni successive) sono quindi fatti scorrere in avanti per la successiva generazione del numero casuale:

```

RNDLP    LDX #4
          LDA SCR,X
          STA SCR +1,X
          DEX
          BPL RNDLP
          RTS
    
```

Il processo è illustrato in Fig. 3.7. Notate che esso implementa uno scorrimento circolare di sette locazioni. Il numero casuale che è stato calcolato viene quindi scritto indietro nella locazione SCR e tutti i valori

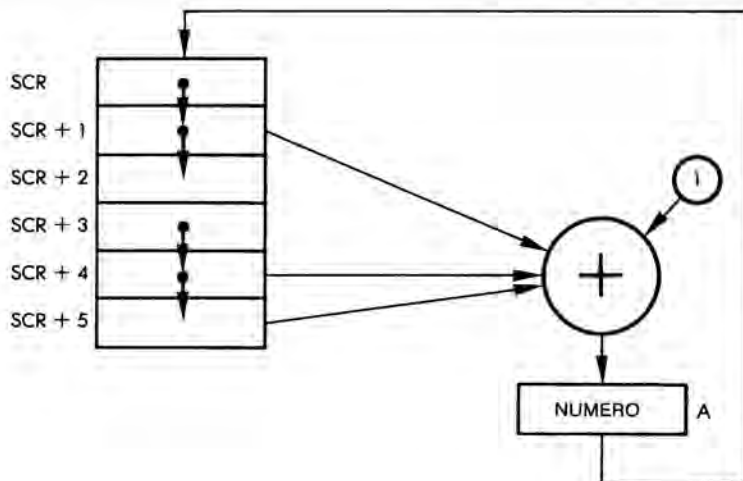


Figura 3.7: Generazione di un numero casuale

precedenti delle locazioni di memoria SCR e successive sono spinti in basso di una posizione. I contenuti precedenti di SCR + 5 vanno persi. Questo assicura che il numero sia ragionevolmente casuale.

## SOMMARIO

Questo gioco coinvolge due giocatori. L'informazione temporale viene conservata per mezzo di cicli annidati. Il numero casuale da indovinare viene generato da un generatore di numeri pseudo-casuali. Per visualizzare il numero binario vien utilizzata una tabella speciale. I LED sulla scheda vengono impiegati per indicare il turno di ciascun giocatore, per visualizzare il numero binario e per indicare il vincitore.

**Esercizio 3-4:** *Cosa succede nel caso in cui tutte le locazioni di memoria da SCR ad SCR + 5 sono inizializzate a zero?*

## CAPITOLO 4

# INDOVINA L'ESADECIMALE

### LE REGOLE

Questo gioco consiste nell'indovinare un numero segreto di due digit (cifre) generato da un computer. Il giocatore fornisce un numero a caso che viene sottoposto al computer e, sulla base della risposta del computer stesso, (indicante il grado di prossimità nel numero enunciato rispetto al numero segreto) si delimita una gamma di numeri a cui appartiene il numero segreto. A questo punto il programma genera un beep, segnalando al giocatore che è pronto per accettare un altro numero. Il programma risponde segnalando il vincitore se il giocatore ha indovinato il numero corretto. Se la risposta del giocatore non è esatta, il programma risponde illuminando i nove LED, indicando la distanza tra il tentativo del giocatore ed il numero corretto. Un LED acceso indica che il numero tentato è molto lontano rispetto al numero corretto, mentre nove LED accesi indicano che il numero è molto vicino a quello da indovinare.

Se il numero è corretto, il programma genera un tono modulato e fa lampeggiare i LED sulla scheda. Al giocatore sono consentiti al massimo dieci tentativi. Se non riesce ad indovinare il numero corretto in dieci tentativi viene emesso un tono basso ed iniziato un nuovo gioco.

### UNA GIOCATA TIPICA

Il beep del computer ci indica che dobbiamo premere un tasto per eseguire un tentativo.

Il nostro tentativo è: "40"

Il computer accende 4 LED

Tentativo successivo: "C0"

Risposta del computer: 3 LED

Tentativo successivo: "20"

Risposta del computer: 3

Siamo abbastanza lontani

Ci siamo allontanati

Il numero deve essere compreso tra C0 e 20

Tentativo successivo: "80"

Risposta del computer: 5

Tentativo successivo: "75"

Risposta del computer: 5

Tentativo successivo: "90"

Risposta: 4

Tentativo successivo: "65"

Risposta: 7

Tentativo successivo: "60"

Risposta: 9

Tentativo successivo: "5F"

Risposta: 8

Tentativo successivo: "61"

Abbiamo vinto !!! Tutti i LED lampeggiano e viene emesso un tono acuto modulato.

Siamo andati più vicini

Non è proprio sotto 80

Siamo fuori strada

Ora siamo vicini

## L'ALGORITMO

La Fig. 4.1 riporta il diagramma di flusso del gioco INDOVINA L'ESADECIMALE. L'algoritmo è immediato:

- viene generato un numero casuale;
- viene caricato un tentativo;
- viene valutata la vicinanza del numero tentato con il numero segreto. Sono disponibili nove livelli di prossimità, ciascuno visualizzato da un LED sulla scheda. Per questo scopo viene utilizzata una tabella di prossimità;
- viene segnalata una vincita o una perdita;
- sono consentiti più tentativi, fino ad un massimo di dieci.

## IL PROGRAMMA

### Strutture dei dati

Il programma è formato da una routine principale, chiamata GETGES e due subroutine, chiamate LITE e TONE. Esso utilizza una semplice struttura dati — una tabella chiamata LIMITS. La Fig. 4.1 riporta il diagramma di flusso, mentre la Fig. 4.2 riporta il listing del programma.

La tabella LIMITS contiene un set di nove valori sui quali verrà verificato il grado di prossimità del tentativo rispetto al numero segreto del computer. Essa è di tipo esponenziale e contiene la sequenza: 1, 2, 4, 8, 16, 32, 64, 128, 200.



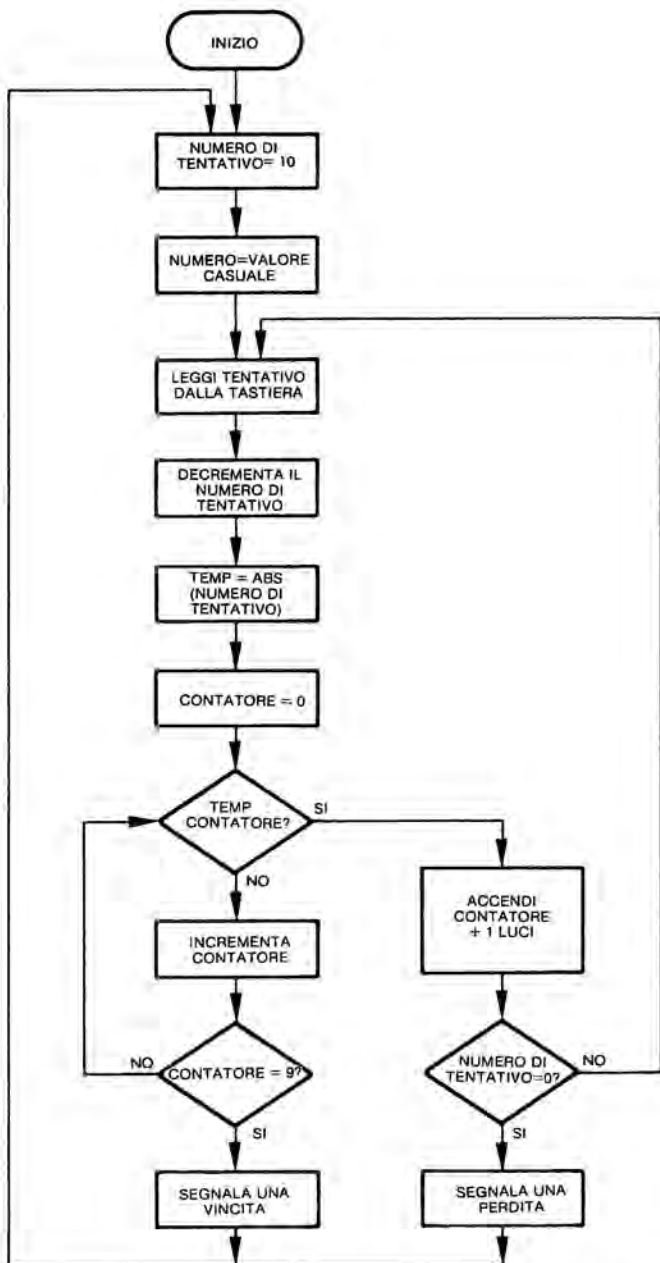


Figura 4.1: Diagramma di flusso per indovina l'esadecimale

## Implementazione del programma

Esaminiamo il programma. Esso risiede all'indirizzo di memoria 200 e non può essere riallocato. Cinque variabili risiedono in pagina zero:

GUESS viene utilizzata per memorizzare il tentativo corrente

GUESS# è il numero del tentativo corrente

DUR e FREQ sono i comuni parametri richiesti per generare un tono (subroutine TONE)

NUMBER è il numero segreto del computer

Come al solito, i registri di direzione dati dei VIA#1 e VIA#3 sono condizionati in modo da pilotare il LED display e leggere la tastiera:

```
LDA #$FF
STA DDRIA  OUTPUT
STA DDRIB  OUTPUT
STA DDR3B  OUTPUT
```

La locazione di memoria DUR viene impiegata per memorizzare la durata del tono da generare mediante la subroutine TONE. Essa viene inizializzata ad "FF" (Esadecimale):

STA DUR

La locazione di memoria GUESS# viene utilizzata per memorizzare il numero di tentativi. Essa è inizializzata a 10:

```
START    LDA #$0A
          STA GUESS#
```

Vengono spenti i LED sulla Scheda Giochi:

```
LDA #00
STA PORTA
STA PORTB
```

Il programma genererà un numero casuale che deve essere indovinato dal giocatore. In questo caso viene ottenuto un numero ragionevolmente casuale leggendo il valore del timer 1, del VIA#1. Esso viene quindi memorizzato all'indirizzo di memoria NUMBER:

```

; 'INDOVINA L'ESADECIMALE'
; GIOCO PER INDOVINARE UN NUMERO ESADECIMALE.
; LO SCOPO DEL GIOCO È INDOVINARE UN NUMERO ESADECIMALE
; GENERATO DAL COMPUTER.
; QUANDO IL COMPUTER ESEGUE UN 'BEEP', SI DOVREBBE
; CARICARE UN TENTATIVO. I TENTATIVI SONO NUMERI
; ESADECIMALI DI DUE DIGIT. QUANDO SONO STATI RICEVUTI
; DUE DIGIT IL COMPUTER VISUALIZZERA' LA VICINANZA
; DEL TENTATIVO ACCENDENDO UN NUMERO DI LED PROPORZIONALE
; ALLA VICINANZA DEL TENTATIVO. SONO CONSENTITI DIECI
; TENTATIVI. SE IL TENTATIVO È CORRETTO IL COMPUTER
; FA LAMPEGGIARE I LED ED ESEGUE UN SUONO
; VARIABILE.
; LA LOCAZIONE D'INGRESSO È $200.

```

```

GETKEY = $100
; INDIRIZZI DEL 6522 VIA # 1:
TIMER = $A004 ; LATCH BASSO DEL TIMER 1
DDR1A = $A003 ; REG. DI DIREZIONE DATI DI PORTA
DDR1B = $A002 ; REG. DI DIREZIONE DATI DI PORTB
PORT1A = $A001 ; PORTA A
PORT1B = $A000 ; PORTA B
; INDIRIZZI DEL 6522 VIA # 3:
DDR3B = $AC02 ; REG. DI DIREZIONE DATI DI PORTB
PORT3B = $AC00 ; PORT B
; STORAGES:
GUESS = $00
GUESS# = $01
DUR = $02
FREQ = $03
NUMBER = $04
; = $200

```

```

0200: A9 FF ; LDA # $FF ; PREDISPONI I REGISTRI DI DIREZIONE
; DATI
0202: 8D 03 A0 ; STA DDR1A
0205: 8D 02 A0 ; STA DDR1B
0208: 8D 02 AC ; STA DDR3B
020B: 85 02 ; STA DUR ; IMPOSTA LE DURATE DEI TONI
020D: A9 0A ; LDA # $0A ; CONSENTITI 10 TENTATIVI
020F: 85 01 ; STA GUESS #
0211: A9 00 ; LDA # 00 ; SPEGNI I LED
0213: 8D 01 A0 ; STA PORT1A
0216: 8D 00 A0 ; STA PORT1B
0219: AD 04 A0 ; LDA TIMER ; ACCETTA UN NUMERO CASUALE PER
; TENTATIVO
021C: 85 04 ; STA NUMBER ; ... E SALVALO.
021E: A9 20 ; GETGES LDA # $20 ; IMPOSTA IL SUONO ACUTO BREVE PER
; SEGNALARE ALL'UTENTE DI CARICARE
; UN TENTATIVO
0220: 20 96 02 ; JSR TONE ; ESEGUI BEEP
0223: 20 00 01 ; JSR GETKEY ; ACCETTA IL TENTATIVO DELL'UTENTE
; DI BASSO ORDINE
0226: 0A ; ASL A ; FALLO SCORRERE NELLA POSIZIONE DI
; ORDINE ELEVATO
0227: 0A ; ASL A
0228: 0A ; ASL A
0229: 0A ; ASL A
022A: 85 00 ; STA GUESS ; SALVALO
022C: 20 00 01 ; JSR GETKEY ; ACCETTA IL TENTATIVO DELL'UTENTE
; DI BASSO ORDINE
022F: 29 0F ; AND # %00001111 ; MASCHERA I BIT DI ORDINE ELEVATO
0231: 05 00 ; ORA GUESS ; AGGIUNGI IL NIBBLE DI ORDINE ELEVATO

```

Figura 4.2: Programma indovina l'esadecimale

0233:	85 00	STA GUESS	:	SALVA IL VALORE FINALE
0235:	A5 04	LDA NUMBER	:	ACCETTA IL NUMERO PER IL CONFRONTO
0237:	38	SEC		
0238:	E5 00	SBC GUESS	:	SOTTRAI IL TENTATIVO DAL NUMERO
			:	PER DETERMINARE LA VICINANZA DEL
			:	TENTATIVO
023A:	B0 05	BCS ALRIGHT	:	VALORE POSITIVO CHE NON NECESSITA
			:	DI AGGIUSTAMENTO
023C:	49 FF	EOR # %11111111	:	RENDI ASSOLUTA LA DISTANZA
023E:	38	SEC	:	ESEGUI IL SUO COMPLEMENTO A DUE
023F:	69 00	ADC # 00	:	... NON UN COMPLEMENTO AD UNO
0241:	A2 00	ALRIGHT LDX # 00	:	IMPOSTA IL CONTATORE DI VICINANZA
0243:	DD AD 02	LOOP CMP LIMITS,X	:	CONFRONTA LA VICINANZA DEL
			:	TENTATIVO CON
			:	LA TABELLA DEI LIMITI PER VEDERE
			:	QUANTE LUCI DEVI ACCENDERE
0246:	B0 27	BCS SIGNAL	:	LA VICINANZA È MOLTO MAGGIORE DEI
			:	LIMITI QUINDI ACCENDI L'INDICATORE
0248:	EB	INX	:	OSSERVA IL SUCCESSIVO LIVELLO DI
			:	VICINANZA
0249:	E0 09	CPX # 9	:	VERIFICA TUTTI I NOVE LIVELLI
024B:	D0 F6	BNE LOOP	:	NO, PROVA IL LIVELLO SUCCESSIVO
024D:	A9 0B	WIN LDA # 11	:	SI: VITTORIA! CARICA IL NUMERO DI
			:	LAMPEGGI
024F:	85 00	STA GUESS	:	USA TENTATIVO COME TEMP
0251:	A9 FF	LDA # \$ FF	:	ACCENDI I LED
0253:	8D 01 A0	STA PORT1A		
0256:	8D 00 A0	STA PORT1B		
0259:	A9 32	WOW LDA # 50	:	VALORE DEL TONO
025B:	20 96 02	JSR TONE	:	ESEGUI IL SEGNALE DI VITTORIA
025E:	A9 FF	LDA # \$ FF		
0260:	AD 01 A0	EOR PORT1A	:	COMPLEMENTA LE PORTE
0263:	8D 01 A0	STA PORT1A		
0266:	8D 00 A0	STA PORT1B		
0269:	C6 00	DEC GUESS	:	ESEGUITI LAMPEGGI/TONI?
026B:	D0 EC	BNE WOW	:	NO, CONTINUA
026D:	F0 9E	BEQ START	:	SI, INIZIA UN NUOVO GIOCO.
026F:	EB	SIGNAL INX	:	INCREMENTA IL LIVELLO DI VICINANZA
			:	CONTATORE IN MODO CHE SIA ACCESO
			:	ALMENO UN LED.
0270:	A9 00	LDA # 0	:	AZZERA LA PORTA DI ORDINE ELEVATO
			:	DEI LED
0272:	8D 00 A0	STA PORT1B		
0275:	20 8E 02	JSR LITE	:	ACCETTA LA STRUTTURA DI LED
0278:	8D 01 A0	STA PORT1A	:	IMPOSTA I LED
027B:	90 05	RCC CC	:	SE CARRY È 1,PB0=1
027D:	A9 01	LDA # 01		
027E:	8D 00 A0	STA PORT1B		
0282:	C6 01	CC DEC GUESS#	:	ESEGUITO UN TENTATIVO
0284:	D0 98	BNE GETGES	:	NON ESATTO, ACCETTA IL SUCCESSIVO.
0286:	A9 BE	LDA # \$ BE	:	SUONO BASSO PER SEGNALARE LA
			:	PERDITA.
0288:	20 96 02	JSR TONE		
028B:	4C 0D 02	JMP START	:	NUOVO GIOCO.
			:	
			:	ROUTINE PER GENERARE LA STRUTTURA DI LED DA ACCENDERE
			:	MEDIANTE SCORRIMENTO A SINISTRA DI UNA STRINGA DI UNI
			:	NELL'ACCUMULATORE
			:	FINCHÉ SI RAGGIUNGE LA POSIZIONE DI BIT CORRISPONDENTE
			:	AL NUMERO IN X.

Figura 4.2: Programma indovina l'esadecimale (continua)

```

028F: A9 00    LITE    LDA # 0      ; AZZERA L'ACCUMULATORE PER
                                ; GENERARE LA STRUTTURA
0290: 38        SHIFT   SEC          ; RENDI ALTO IL BIT BASSO.
0291: 2A        ROL A    ; FALLO SCORRERE IN
0292: CA        DEX      ; UN BIT, FATTO ...
0293: D0 FB    BNE SHIFT ; RICICLA SE NON FATTO.
0295: 60        RTS      ; RITORNO

; ROUTINE PER LA GENERAZIONE DEL SUONO.
;
0296: 85 03    TONE    STA  FREQ
0298: A9 00    LDA # $ 00
029A: A6 02    LDX  DUR
029C: A4 03    LDY  FREQ
029E: 88      FL2    DEY
029F: 18      FL1    CLC
02A0: 90 00    BCC  + 2
02A2: D0 FA    BNE  FL1
02A4: 49 FF    EOR  # $ FF
02A6: 8B 00 AC STA  PORT3B
02A8: CA      DEX
02AA: D0 FD    BNE  FL2
02AC: 60      RTS

; TABELLA DEI LIMITI PER I LIVELLI DI VICINANZA.
;
02AD: C8      LIMITS .BYTE 200,128,64,32,16,8,4,2,1
02AE: 80
02AF: 40
02B0: 20
02B1: 10
02B2: 08
02B3: 04
02B4: 02
02B5: 01

SYMBOL TABLE:
GETKEY      0100      TIMER      A004      DDR1A      A003
DDR1B       A002      PORT1A     A001      PORT1B     A000
DDR3B       AC02      PORT3B     AC00      GUESS      0000
GUESS#      0001      DUR        0002      FREQ       0003
NUMBER      0004      START      020D      GETGES     021E
ALRIGHT     0241      LOOP       0243      WIN        024D
WOW         0259      SIGNAL     026F      CC         0282
LITE        028E      SHIFT      0290      TONE       0296
FL2         029C      FL1        029E      LIMITS     02AD

```

Figura 4.2: Programma indovina l'esadecimale (continua)

LDA TIMER  
STA NUMBER

Latch del timer 1

Non è richiesto un generatore di numeri casuali in quanto le richieste di numeri casuali si verificano ad intervalli di tempo casuali, contrariamen-

00	ORB (DA PB0 A PB7)	Porta A, Dati di I/O
01	ORA (DA PA0 A PA7)	Impiegato per il controllo - Influenza handshake
02	DDR B	Registri di direzione dati
03	DDR A	
04	T1L-L/T1C-L	Contatore basso
05	T1C-H	Contatore alto
06	T1L-L	Latch basso
07	T1L-H	Latch alto
08	T2L-L/T2C-L	Contatore basso
09	T2C-H	Contatore alto
0A	SR	Registro di scorrimento
0B	ACR	Ausiliari
0C	PCR (CA1,CA2,CB2,CB1)	Periferiche
0D	IFR	Flag
0E	IER	Abilitazione
0F	ORA	Uscita registro A (non influenza handshake)

Figura 4.3: Mappa di memoria del 6522 VIA

te alla maggior parte degli altri giochi descritti. Un'osservazione importante sull'impiego di T1CL di un 6522 VIA è il fatto che è spesso chiamato "latch" mentre in realtà si tratta di un "contatore" che esegue un'operazione di lettura! I suoi contenuti *non* sono congelati durante un'operazione di lettura, come nel caso di un latch. Essi sono decrementati continuamente. Quando si decrementa a zero, il contatore viene ricaricato da un latch "reale".

Si noti che in Fig. 4.3 T1L-L appare due volte - agli indirizzi 04 e 06. Questo può essere causa di confusione ed occorre un ulteriore chiarimento. La locazione 4 corrisponde al contatore; la locazione 6 corrisponde al latch. In questo caso viene letta la locazione 4.

Siamo pronti a procedere. Viene generato un tono acuto per segnalare al giocatore che può fare un tentativo. La durata della nota è memorizza-

ta alla locazione di memoria DUR mentre la frequenza del tono è settata dai contenuti dell'accumulatore:

```
GETGES      LDA #$20      Tono acuto
            JSR TONE
```

Per ogni tentativo occorre memorizzare due valori di tasto. Per ottenere il numero del tasto premuto viene impiegata la subroutine GETKEY, che lo memorizza nell'accumulatore. Una volta ottenuto il primo carattere, viene fatto scorrere a sinistra di quattro posizioni, nella posizione del nibble di ordine elevato ed ottenuto il carattere successivo. (Vedere Fig. 4.4).

```
JSR GETKEY
ASL A
ASL A
ASL A
ASL A
STA GUESS
JSR GETKEY
```

Una volta che il secondo carattere è stato trasferito nell'accumulatore, il carattere precedente, che è stato salvato nella locazione di memoria

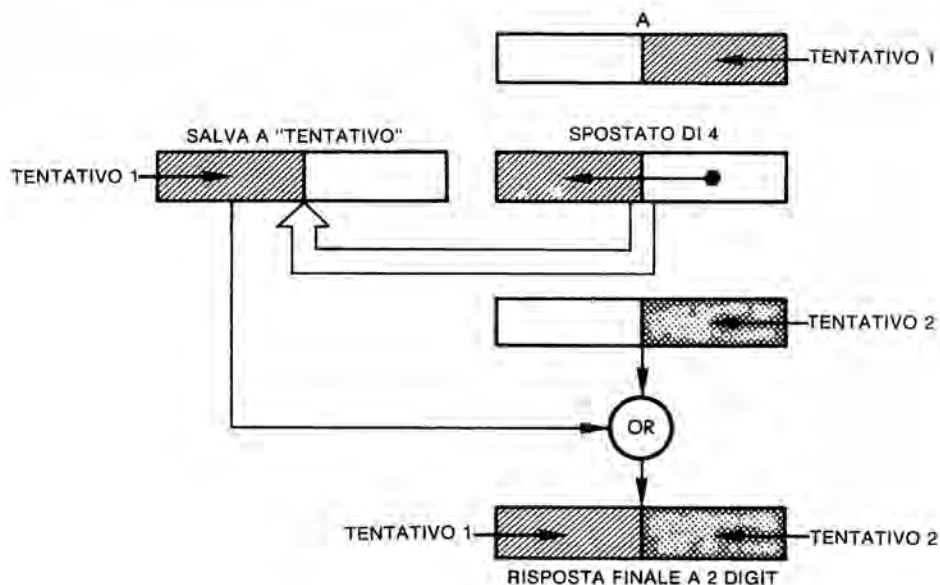


Figura 4.4: Raccolta dei tentativi del giocatore

GUESS, viene messo in OR con i contenuti dell'accumulatore:

```
AND #%00001111
ORA GUESS
```

E quindi ri-memorizzato alla locazione di memoria GUESS:

```
STA GUESS
```

Notate che, eseguito il tentativo, deve essere confrontato con il numero casuale memorizzato dal computer alla locazione di memoria NUMBER. Viene quindi eseguita una sottrazione:

```
LDA NUMBER
SEC
SBC GUESS
```

Se la differenza è negativa, deve essere completata:

BCS ALRIGHT	Positiva?
EOR #%11111111	È negativa: esegui il complemento
SEC	Esegui il complemento a due
ADC #00	Somma uno

Una volta calcolata la “distanza” del tentativo dal numero reale, occorre impostare il “contatore di prossimità” ad un valore tra 1 e 9 (vengono utilizzati solo nove LED). Questo viene eseguito da un ciclo che confronta la “distanza” assoluta del tentativo dal numero corretto con il valore di gruppo nella tabella LIMITS. Il numero del valore di gruppo corretto diviene il valore assegnato alla prossimità del numero di tentativo al numero segreto. Il registro indice X viene inizializzato a 0, la ricerca dei valori di gruppo viene eseguita con il modo di indirizzamento indicizzato. Vengono eseguiti i confronti finché la distanza è minore del valore di gruppo, oppure finché X supera 9; cioè fino a consultare il valore più elevato nella tabella.

ALRIGHT	LDX #00	
LOOP	CMP LIMITS, X	Valore limite di consultazione
	BCS SIGNAL	
	INX	Minore prossimità
	CPX #9	Continua fino a 10 tentativi
	BNE LOOP	



A questo punto, quando si verifica una ramificazione a SIGNAL, la distanza tra il tentativo ed il numero reale è 0: è una vittoria. Questo è segnalato dal lampeggio dei LED e dalla generazione di uno speciale tono:

WIN	LDA #11 STA GUESS LDA #FF STA PORT1A STA PORT1B	Memoria scratch
WOW	LDA #50 JSR TONE	Passo del tono Generazione del tono

Il lampeggio viene generato complementando ripetutamente i LED:

LDA #\$FF EOR PORT1A STA PORT1A STA PORT1B	Complementa le porte
---	----------------------

Il ciclo viene eseguito ancora:

```
DEC GUESS
BNE WOW
```

Infine, quando l'indice del ciclo (GUESS) diventa zero, si verifica una ramificazione indietro all'inizio del programma principale: START:

```
BEQ START
```

Comunque, se il tentativo corrente non è corretto, si verifica una ramificazione a SIGNAL durante il confronto di gruppo, con i contenuti del registro X uguali al valore di prossimità: cioè il numero di LED da accendere. In dipendenza della vicinanza del tentativo al numero segreto, saranno accesi da 1 a 9 LED:

SIGNAL	INX	Incrementa il livello di vicinanza
	LDA #0	Azzera la porta di ordine elevato dei LED
	STA PORT1B JSR LITE STA PORT1A	Accetta la struttura di LED

```

BCC CC
LDA #01
STA PORT1B

```

Se carry è 1, PBO = 1

Il numero di LED da accendere è contenuto in X. Occorre quindi convertire questo numero in un pattern corretto da inviare alla porta d'uscita. Questo viene eseguito dalla subroutine LITE, descritta più avanti.

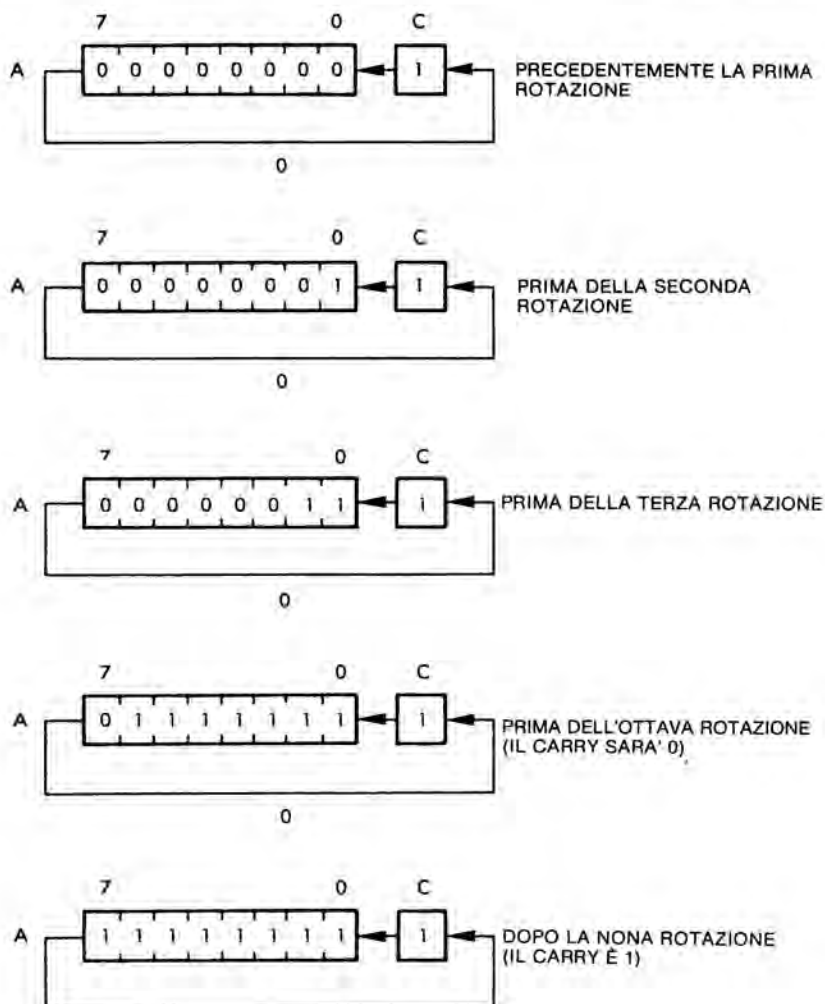


Figura 4.5: Realizzazione della struttura di LED per gli 8 LED

Se devono essere accessi 9 LED, LITE pone ad 1 il bit carry (riporto). Viene eseguito un test esplicito sul carry (il pattern 01 viene quindi inviato a PORT1B). Successivamente viene decrementato il numero corrente di tentativi. Se è zero, il giocatore ha perso: viene generato il segnale di sconfitta ed inizia un nuovo gioco; altrimenti si ha il tentativo successivo:

CC	DEC GUESS #	
	BNE GETGES	Tentativo errato?
	LDA #\$BE	Suono basso
	JSR TONE	
	JMP START	Nuovo gioco

## Le subroutine

### Subroutine LITE

La subroutine LITE genererà il pattern richiesto per illuminare da 1 ad 8 LED, in dipendenza del numero contenuto nel registro X. I bit "1" richiesti sono fatti scorrere a destra nell'accumulatore ad ogni decremento del registro X. La Fig. 4.5 riporta un esempio.

All'uscita dalla subroutine, l'accumulatore contiene il pattern corretto richiesto per accendere il numero di LED specificati. Se si devono accendere nove LED, il pattern sarebbe formato da tutti uni ed il bit carry dovrebbe essere posto ad 1:

LITE	LDA #0	
SHIFT	SEC	"1" di partenza
	ROL A	Ruota l'"1" in posizione
	DEX	Fatto
	BNE SHIFT	
	RTS	

### Subroutine TONE

La subroutine TONE genererà un tono della durata specificata dalla costante nella locazione di memoria DUR, alla frequenza specificata dai contenuti dell'accumulatore. Il registro indice Y vien impiegato come contatore del ciclo interno. Come al solito, il tono viene generato commutando on ed off l'altoparlante connesso a PORT3B con un periodo di tempo opportuno:

TONE	STA FREQ
	LDA #\$00
	LDX DUR
FL2	LDY FREQ
FL1	DEY
	CLC
	BCC . + 2
	BNE FL1
	EOR #\$FF
	STA PORT3B
	DEX
	BNE
	RTS

## SOMMARIO

In questo caso come generatore di numeri casuali viene impiegato un timer latch (cioè un registro hardware), anzichè una routine software. Per visualizzare un valore viene utilizzata una semplice routine "LITE", mentre, per generare il suono viene impiegata la solita routine TONE.

## ESERCIZI

**Esercizio 4-1:** *Migliorate il programma "Indovina l'esadecimale" aggiungendo la seguente caratteristica. Alla fine di ogni gioco, se il giocatore ha perso, il programma visualizza il numero che si doveva indovinare per circa tre secondi, prima di iniziare un nuovo gioco.*

**Esercizio 4-2:** *Cosa succederebbe se non fosse presente SEC alla locazione 290 esadecimale?*

**Esercizio 4-3:** *Quali sono i vantaggi e gli svantaggi dell'impiego del valore del timer per generare un numero casuale? Cosa succede dei numeri successivi? Saranno correlati? Identici?*

**Esercizio 4-4:** *Quante volte il programma precedente fa lampeggiare i LED per indicare una vittoria?*

**Esercizio 4-5:** *Esaminare la routine WIN (linea 24D). In caso di vittoria il tono verrà eseguito una o più volte?*

**Esercizio 4-6:** *Qual'è lo scopo delle due istruzioni presenti agli indirizzi 29F e 2A0? (Suggerimento: leggete il Capitolo 2).*

**Esercizio 4-7:** *Il programma dovrebbe inizializzare il timer?*

**Esercizio 4-8:** *Il numero di LED accesi in risposta ad un tentativo è linearmente correlato alla vicinanza rispetto al valore corretto?*

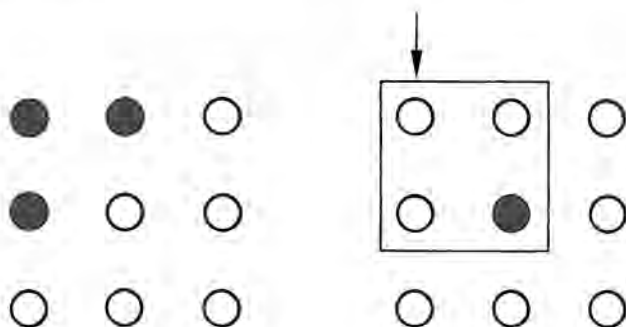
## CAPITOLO 5

# QUADRATO MAGICO

### LE REGOLE

Il gioco consiste nell'illuminare un quadrato perfetto sulla scheda, cioè i LED 1, 2, 3, 6, 9, 8, 7 e 4 ma non il LED #5 al centro.

Il gioco inizia con un pattern casuale. Il giocatore può modificare il LED pattern sulla scheda impiegando una tastiera, in quanto ogni tasto complementa un gruppo di LED. Per esempio, ogni tasto corrispondente alle posizioni d'angolo dei LED (numero dei tasti: 1, 3, 9 e 7) complementa il pattern del quadrato a cui è associato. Il tasto #1 complementerà il pattern del quadrato formato dai LED 1, 2, 4 e 5. Assumendo che i LED 1, 2 e 4 siano accesi, premendo il tasto #1 si otterrà il pattern seguente: 1-off, 2-off, 4-off, 5-on.

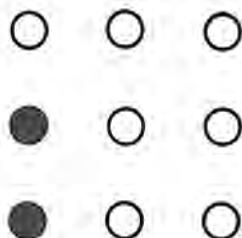


Quindi il pattern formato dai LED 1, 2, 4 e 5 è stato complementato e solo il LED #5 è acceso dopo la pressione del tasto #1. Premendo ancora il tasto #1 si avrà: 1, 2 e 4-on con 5-off. Premendo un tasto due volte si ottengono due operazioni successive di complemento e quindi, globalmente, non si ha alcun effetto.

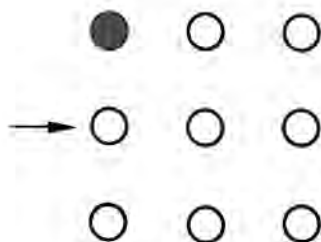
Analogamente, il tasto #9 complementa il quadrato in basso a destra formato dai LED 5, 6, 8 e 9.

Il tasto #3 complementa il pattern formato dai LED 2, 3, 5 e 6. Il tasto #7 complementa il pattern formato dai LED 4, 5, 7 ed 8.

I tasti di bordo corrispondenti ai LED 2, 4, 6 ed 8 complementano il pattern formato dai tre LED del lato esterno di cui essi sono al centro. Per esempio, premendo il tasto #2 si completerà il pattern dei LED 1, 2 e 3. Assumiamo un pattern iniziale con accesi i LED 1, 2 e 3. Premendo il tasto #2 si otterrà il complemento di tale pattern e quindi lo spegnimento di tutti i LED. Analogamente, assumiamo un pattern iniziale con accessi i LED del bordo verticale sinistro 4 e 7;



Premendo il tasto #4 si otterrà un pattern in cui è acceso il LED #1 e sono spenti i LED 4 e 7.

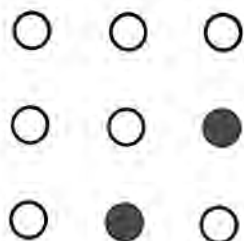


È STATO PREMUTO IL TASTO 4

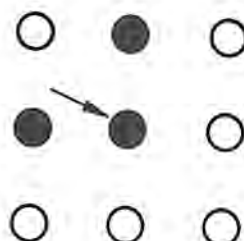
Analogamente il tasto #8 completerà il pattern formato dai LED 7, 8 e 9, mentre il tasto #6 completerà il pattern formato dai LED 3, 6 e 9.

Infine, premendo il tasto #5 (posizione del LED centrale) si eseguirà il complemento del pattern formato dai LED 2, 4, 5, 6 ed 8. Per esempio,

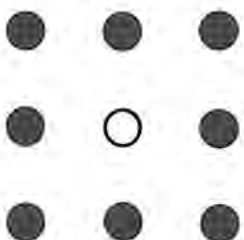
assumiamo il seguente pattern iniziale in cui sono accesi solo i LED 6 ed 8:



Premendo il tasto #5 si otterrà l'accensione dei LED 2, 4 e 5:



La combinazione vincente in cui sono accesi tutti i LED ai bordi dei quadrati si ottiene premendo una sequenza opportuna di tasti.



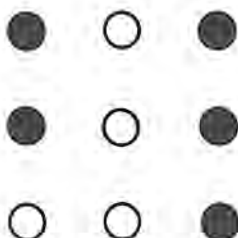
La dimostrazione matematica che è sempre possibile trovare una combinazione vincente è lasciata al lettore come esercizio. Il programma conferma che il giocatore ha ottenuto il pattern vincente facendo lampeggiare i LED.

Per iniziare un nuovo gioco occorre premere il tasto "0". A questo punto verrà visualizzato sulla scheda un nuovo pattern casuale. Gli altri tasti vengono ignorati.

### UNA GIOCATA TIPICA

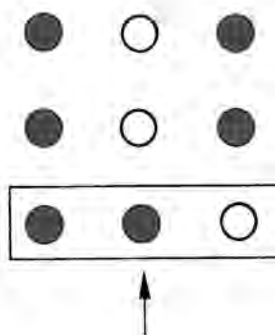
Ecco una sequenza tipica:

Il pattern iniziale è: 1 - 3 - 4 - 6 - 9



Mossa: premere il tasto #8.

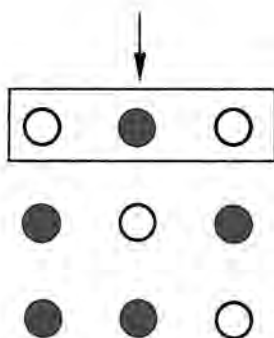
Il pattern risultante è: 1-3-4-6-7-8



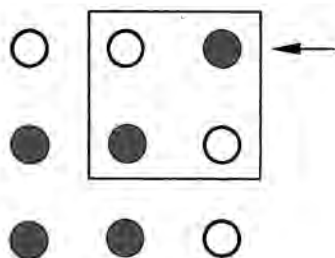
Mossa successiva: premere il tasto #2.

Il pattern risultante è: 2-4-6-7-8.

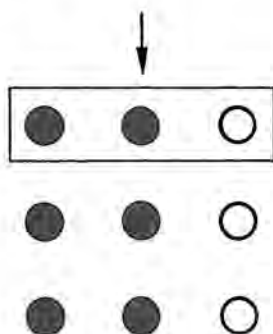




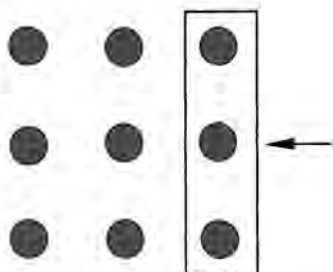
Mossa successiva: premere il tasto #3.  
Il pattern risultante è: 3-4-5-7-8.



Mossa successiva: premere il tasto #2.  
Il pattern risultante è: 1-2-4-5-7-8.



Mossa successiva: premere il tasto #6.  
Il pattern risultante è: 1-2-3-4-5-6-7-8-9.



Notate che questo pattern ha tutti i LED della scheda accesi. Non è la situazione vincente in quanto il LED #5 dovrebbe essere off. Procediamo.

Mossa successiva: la fine del gioco è lasciata al talento matematico del lettore. Lo scopo principale era quello di dimostrare l'effetto delle varie mosse.

Suggerimento: una possibile sequenza vincente è: 2-4-6-8-5!

Regola generale: per arrivare più facilmente alla vittoria in questo gioco occorre cercare di arrivare velocemente ad un pattern simmetrico sulla scheda. Una volta ottenuto un pattern simmetrico è relativamente semplice arrivare ad ottenere un quadrato perfetto. Generalmente si ottiene un pattern simmetrico premendo i tasti corrispondenti ai LED che sono off nella scheda ma che dovrebbero essere on per completare il pattern.

## L'ALGORITMO

Sulla scheda viene generato un pattern impiegando dei numeri casuali. Viene quindi identificato il tasto corrispondente alla mossa del giocatore e viene complementato il gruppo di LED corrispondente.

Per specificare i LED che formano il gruppo relativo ad ogni tasto occorre utilizzare una tabella.

Si verifica quindi se il nuovo pattern ottenuto è un quadrato perfetto. Se sì, il giocatore vince. In caso contrario il processo ricomincia.

La Fig. 5.1 riporta in modo dettagliato il diagramma di flusso.

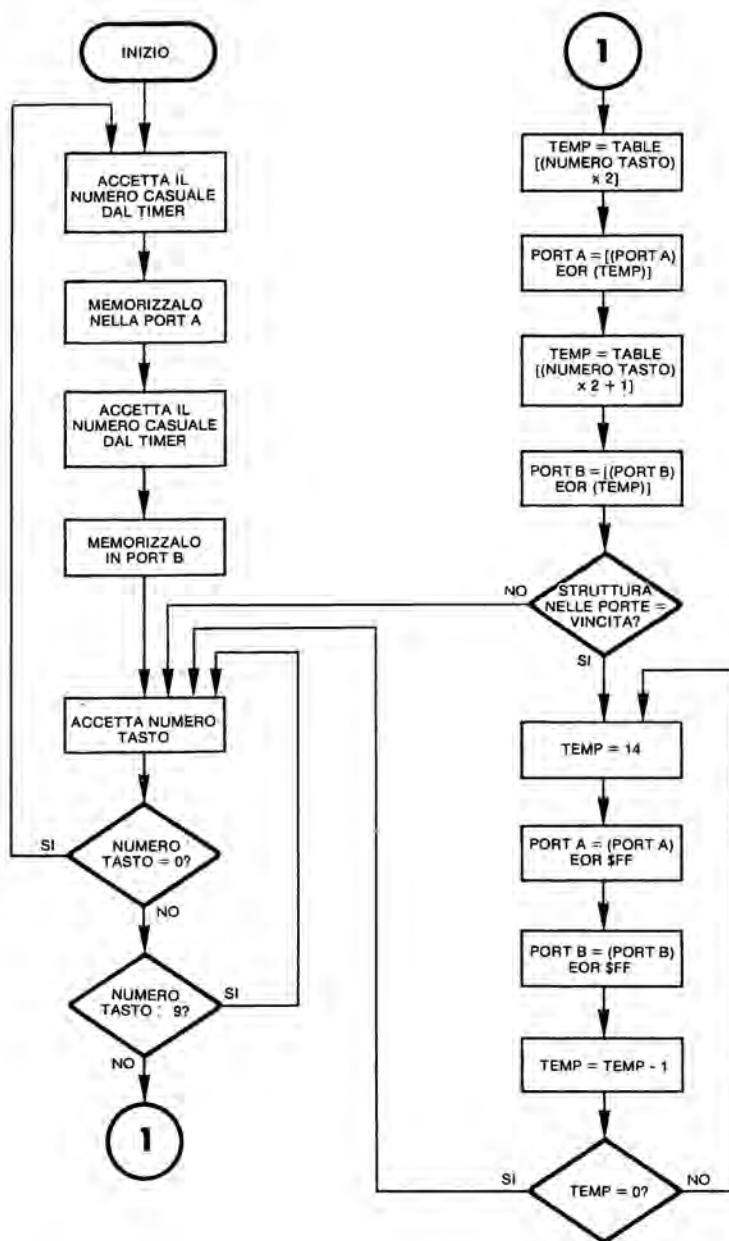


Figura 5.1: Diagramma di flusso del quadrato magico

## IL PROGRAMMA

### Strutture dei dati

In questo caso il problema principale è escogitare un modo efficiente per complementare il pattern di LED corretto in corrispondenza della pressione di un tasto. L'operazione di complemento può essere eseguita mediante un'istruzione di OR esclusivo. In questo caso, il pattern impiegato con l'istruzione di EOR dovrebbe contenere un "1" in ogni posizione di LED che deve essere complementata ed uno "0" in tutte le altre posizioni. La soluzione è abbastanza semplice: viene impiegata una tabella a nove ingressi, chiamata TABLE. Ogni ingresso della tabella corrisponde ad un tasto ed ha 16 bit di cui solo 9 bit contengono un "1", nella posizione corretta, indicando il LED che sarà influenzato dal tasto.

Per esempio, abbiamo visto che il tasto 1 risulterà complementando i LED 1, 2, 4 e 5. L'ingresso corrisponde alla tabella è perciò 0, 0, 0, 1, 1, 0, 1, 1, 0, dove i bit 1, 2, 4 e 5 (iniziando la numerazione da 1 come con i tasti) sono stati posti ad "1". Più precisamente, utilizzando una struttura a 16 bit, si ha:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1

La tabella completa è riportata in Fig. 5.2.

TASTO	STRUTTURA	
1	00011011	00000000
2	00000111	00000000
3	00110110	00000000
4	01001001	00000000
5	10111010	00000000
6	00100100	00000001
7	11011000	00000000
8	11000000	00000001
9	10110000	00000001

Figura 5.2: Tabella di complementazione

### Implementazione del programma

All'inizio del gioco deve essere attivata sulla scheda una struttura casuale di LED accesi. Questo, viene eseguito, analogamente al capitolo

```

PROGRAMMA 'QUADRATO MAGICO'
OGNI TASTO 1-9 DELLA TASTIERA ESADECIMALE È ASSOCIATO
CON UN LED DI UNA MATRICE 3x3. QUANDO VIENE PREMUTO UN
TASTO, CAMBIA LA STRUTTURA DEI LED ACCESI NELLA MATRICE.
LO SCOPO DEL GIOCO È DI CONVERTIRE LA STRUTTURA CAUSALE
CON CUI INIZIA IN UN QUADRATO DI LED ACCESI
PREMENDO I TASTI, I LED LAMPEGGERANNO QUANDO
SI OTTIENE LA STRUTTURA VINCENTE.
IL TASTO # 0 PUO' ESSERE UTILIZZATO IN QUALSIASI MOMENTO
PER RI-INIZIALIZZARE IL GIOCO CON UNA NUOVA STRUTTURA.

```

```

GETKEY = $100
T1CL = $A004 ; REGISTRO BASSO DL TIMER NEL 6522 VIA
PORT1 = $A001 ; PORTA A DEL 6522 VIA
PORT2 = $A000 ; PORTA B DEL 6522 VIA
TEMP = $0000 ; MEMORIA TEMPORANEA
DDRA = $A003 ; REGISTRO DI DIREZIONE DATI DELLA
; PORTA A
DDRB = $A002 ; ANGOLO PER LA PORTA B
; = $200

```

```

COMMENTI: QUESTO PROGRAMMA USA UN REGISTRO TIMER COME
SORGENTE DI NUMERI CASUALI. SE NON DISPONIBILE, SI
POTREBBE IMPIEGARE UN GENERATORE DI NUMERI CASUALI.
PERO', A CAUSA DELLA SUA RIPETIBILITA', POTREBBE NON OPE-
RARE
BENE. QUESTO PROGRAMMA USA I REGISTRI DELLA PORTA
A PER MEMORIZZARE LA STRUTTURA DI LED. POICHÉ IL
PROCESSORE LEGGE LA POLARITA' DELLE LINEE D'USCITA,
UN CARICO ECCESSIVO SULLE LINEE POTREBBE IMPEDIRE
UN CORRETTO FUNZIONAMENTO DEL PROGRAMMA.

```

```

0200: A9 FF          LDA #$FF          ; CONFIGURA LE PORTE COME USCITE
0202: 8D 03 A0       STA DDRA
0205: 8D 02 A0       STA DDRB
0208: AD 04 A0 START LDA T1CL          ; ACCETTA IL PRIMO NUMERO CASUALE
020B: 8D 01 A0       STA PORT1
020F: AD 04 A0       LDA T1CL          ; ... ED IL SECONDO.
0211: 29 01          AND #01          ; MASCHERA LA RIGA IN BASSO DI LED
0213: 8D 00 A0       STA PORT2
0216: 20 00 01 KEY   JSR GETKEY
0219: C9 00          CMP #0          ; IL TASTO DEVE ESSERE 1-9: È 0?
021B: F0 EB          BEQ START        ; SÌ, RI-INIZIALIZZA IL GIOCO CON NUOVA
; STRUTTURA.
021D: C9 0A          CMP #10         ; È MINORE DI 10?
021F: 10 F5          BPL KEY          ; + SE TASTO = 10, QUINDI ACCETTANE
; UN ALTRO

```

```

DI SEGUITO VIENE IMPIEGATO IL NUMERO DI TASTO COME INDICE
PER TROVARE NELLA TABELLA UNA STRUTTURA DI BIT IMPIEGATA
PER IL COMPLEMENTO DEI LED

```

```

0221: 38             SEC              ; DECREMENTA A PER L'ACCESSO ALLA
; TABELLA
0222: E9 01          SBC #1
0224: 0A             ASL A            ; MOLTIPLICA Ax2, POICHÉ OGNI
; INGRESSO DELLA
; TABELLA È DI DUE BYTE.
0225: AA             TAX
0226: AD 01 A0       LDA PORT1        ; USA A COME INDICE
; ACCETTA I CONTENUTI DELLA PORTA
; PER IL COMPLEMENTO

```

Figura 5.3: Programma del quadrato magico

```

0229: 5D 6B 02      EOR TABLE, X      ; OR ESCLUSIVO DEI CONTENUTI PORTA E
                                ; STRUTTURA
022C: 8D 01 A0      STA PORT1      ; RIPRISTINA PORT1
022F: AD 00 A0      LDA PORT2      ; RIPETI PER PORT2,
0232: 5D 6C 02      EOR TABLE+1, X ; USANDO L'INGRESSO SUCCESSIVO
                                ; ALLA TABELLA.
0235: 29 01          AND #01      ; MASCHERA I LED DELLA RIGA IN BASSO
0237: 8D 00 A0      STA PORT2      ; ... E RIPRISTINA.

; QUESTA PARTE CONTROLLA I LED PER UNA SITUAZIONE VINCENTE
;
023A: 4A            LSR A          ; SCORRIMENTO DEL BIT O DELLA PORT1
                                ; NEL CARRY.
023B: 90 D9          BCC KEY      ; SE NON STRUTTURA VINCENTE,
                                ; ACCETTA MOSSA SUCCESSIVA
023D: AD 01 A0      LDA PORT1      ; CARICA PORT1 CON TEST DI VITTORIA
0240: C9 FF          CMP #%11101111 ; CONTROLLA SE STRUTTURA VINCENTE
0242: D0 D2          BNE KEY      ; NESSUNA VITTORIA, ACCETTA LA
                                ; MOSSA SUCCESSIVA

; VITTORIA: LAMPEGGIA 4 VOLTE I LED OGNI 1/2 SECONDO
;
0244: A9 0E          LDA #14
0246: 85 00          STA TEMP      ; CARICA IL NUMERO DI LAMPEGGI
024B: A2 20          BLINK LDX #$20 ; COSTANTE DI RITARDO PER 0.08 SEC
024A: A0 FF          DELAY LDY #$FF ; CICLO ESTERNO DELLA ROUTINE DI
                                ; RITARDO VARIABILE,
                                ; IL CUI TEMPO DI RITARDO È
                                ; 2556 * (CONTENUTI) DI X ALLA
                                ; CHIAMATA)
024C: EA            DLY NOP        ; CICLO DI 10 MICROSEC
024D: D0 00          BNE .+2
024F: 88            DEY
0250: D0 FA          BNE DLY
0252: CA            DEX
0253: D0 F5          BNE DELAY
0255: AD 01 A0      LDA PORT1      ; ACCETTA LE PORTEE COMPLEMENTALE
0258: 49 FF          EOR #$FF
025A: 8D 01 A0      STA PORT1
025D: AD 00 A0      LDA PORT2
0260: 49 01          EOR #1
0262: 8D 00 A0      STA PORT2
0265: C6 00          DEC TEMP      ; CONTO ALLA ROVESCIA DEL NUMERO DI
                                ; LAMPEGGI
0267: D0 DF          BNE BLINK      ; RIPETI SE NON FATTO
0269: F0 AB          BEQ KEY      ; ACCETTA LA MOSSA SUCCESSIVA

; TABELLA DEI CODICI IMPIEGATI PER COMPLEMENTARE I LED
;
026B: 1B            TABLE .BYT %00011011,%00000000
026C: 00            .BYT %00000111,%00000000
026D: 07            .BYT %00110110,%00000000
026E: 00            .BYT %01001001,%00000000
026F: 36            .BYT %10111010,%00000000
0270: 00            .BYT %00100100,%00000001
0271: 49            .BYT %01001001,%00000000
0272: 00            .BYT %10111010,%00000000
0273: BA            .BYT %00100100,%00000001
0274: 00
0275: 24
0276: 01

```

Figura 5.3: Programma del quadrato magico (continua)

```

0277: D8                      .BYT %11011000,%00000000
0278: 00
0279: C0                      .BYT %11000000,%00000001
027A: 01
027B: B0                      .BYT %10110000,%00000001
027C: 01

```

#### SYMBOL TABLE:

GETKEY	0100	T1CL	A004	PORT1	A001
PORT2	A000	TEMP	0000	DDRA	A003
DDRB	A002	START	0208	KEY	0216
BLINK	0248	DELAY	024A	DLY	024C
TABLE	026B				

⌘

Figura 5.3: Programma del quadrato magico (continua)

precedente, leggendo il valore del VIA #1 timer. Qualora non sia disponibile un timer, si dovrebbe utilizzare una routine per la generazione di numeri casuali.

I registri di direzione dati delle Porte A e B del VIA sono configurati come uscite per pilotare i LED:

```

LDA #$FF
STA DDRA
STA DDRB

```

I numeri “casuali” sono quindi ottenuti leggendo il valore del timer 1 del VIA e sono utilizzati per fornire una struttura casuale dei LED. (Due numeri forniscono 16 bit, di cui ne sono utilizzati 9).

START	LDA T1CL	Accetta il primo numero
	STA PORT1	Impiegalo
	LDA T1CL	Accetta il secondo numero
	AND #01	Conserva solo la posizione 0
	STA PORT2	Usala

Al capitolo precedente è stata fornita una spiegazione dell'impiego di T1CL. Successivamente il programma esamina la tastiera per vedere quale tasto ha premuto il giocatore. Esso accetterà solo gli ingressi da “0” a “9” e scarterà tutti gli altri:

```

KEY      JSR GETKEY

```

CMP #0	È il tasto 0?
BEQ START	
CMP #10	
BPL KEY	Se tasto = 10 accettane un altro

Se il giocatore ha premuto il tasto "0", il programma viene ri-inizializzato con una nuova struttura dei LED del display. Se il valore premuto è compreso tra "1" e "9", allora viene eseguita la corrispondente variazione sulla configurazione LED. Il numero del tasto verrà utilizzato come indice alla tabella dei codici di complementazione. Poichè i tasti hanno i numeri compresi tra 1 e 9, per l'impiego come indice occorre preliminarmente decrementare il numero di tasto di un'unità. Inoltre, poichè la tabella contiene ingressi di due byte, il numero indice deve essere moltiplicato per 2. Questo viene eseguito dalle seguenti tre istruzioni:

SEC	
SBC #1	Sottrai 1
ASL A	Moltiplica per 2

Si ricordi che uno scorrimento a sinistra è equivalente, nel sistema binario, ad una moltiplicazione per 2. Il valore risultante viene impiegato come indice e memorizzato nel registro indice X:

### TAX

La struttura dei LED viene memorizzata nei registri dei dati della Porta A. Verrà eseguita la complementazione mediante l'istruzione EOR sulla Porta 1 e quindi sulla Porta 2:

LDA PORT1	
EOR TABLE,X	Complementa la porta 1
STA PORT1	
LDA PORT2	Analogamente per porta 2
EOR TABLE+1,X	
AND #01	Maschera i bit non utilizzati
STA PORT2	

Notate che il valore aritmetico del tempo di assemblaggio viene impiegato per specificare il secondo byte della tabella:

EOR TABLE + 1,X



Una volta che la struttura è stata complementata, il programma controlla se si tratta di un pattern vincente. A tale scopo, vengono confrontati i contenuti della Porta 1 e della Porta 2 con la struttura corretta. Per la Porta 2 si ha: "0, 0, 0, 0, 0, 0, 0, 1", mentre per la Porta 1 si ha: "1, 1, 1, 0, 1, 1, 1, 1". Si può eseguire immediatamente il test del bit 0 della Porta 2 dopo uno scorrimento a sinistra:

LSR A	Fai scorrere il bit 0 della Porta 2
BCC KEY	

I contenuti della Porta 1 devono essere esplicitamente confrontati con la struttura corretta:

```

LDA PORT1
CMP #%11101111
BNE KEY

```

Per confermare una vincita, i LED vengono fatti lampeggiare sulla scheda. TEMP viene utilizzato come variabile contatore; X viene utilizzato per impostare la durata del ritardo fisso. Y viene utilizzato come contatore del ciclo più interno. Ogni porta viene complementata dopo che è trascorso il ritardo.

	LDA #14	
	STA TEMP	Carica il numero di lampeggi
BLINK	LDX #\$20	Costante di ritardo per 0,08sec
DELAY	LDY #\$FF	Ciclo esterno della routine
		di ritardo variabile, il cui
		tempo di ritardo è 2556x
		(Contenuti di X alla chiamata)
		ciclo di 10 µs
DLY	NOP	
	BNE. + 2	
	DEY	
	BNE DLY	
	DEX	
	BNE DELAY	
	LDA PORT1	Accetta le porte
		e complementale
	EOR #\$FF	
	STA PORT1	
	LDA PORT2	

EOR #1  
STA PORT2  
DEC TEMP

BNE BLINK  
BEQ KEY

Conta alla rovescia il  
numero di lampeggi  
Ripeti se non è fatto  
Accetta l'ingresso successivo

## SOMMARIO

Questo gioco di abilità richiede una tabella speciale per le diverse complementazioni. Il timer viene utilizzato direttamente per fornire un numero pseudo-casuale, al posto del programma. La configurazione LED viene memorizzata direttamente sui registri del chip di I/O.

## ESERCIZI

**Esercizio 5-1:** *Riscrivete la fine del programma utilizzando una subroutine di ritardo.*

**Esercizio 5-2:** *La struttura di partenza sarà ragionevolmente casuale?*

**Esercizio 5-3:** *Aggiungete degli effetti sonori.*

**Esercizio 5-4:** *Consentite l'impiego del tasto "A" per eseguire una diversa variazione, quale una complementazione totale.*

**Esercizio 5-5:** *(più difficile) Scrivete un programma che consenta al computer di giocare e vincere.*

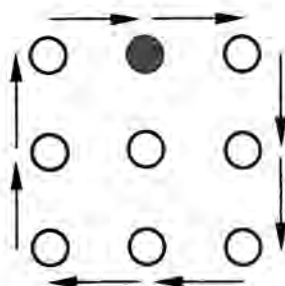
**Esercizio 5-6:** *Aggiungete all'esercizio precedente la seguente caratteristica: registrate il numero di mosse eseguite dal computer e giocate contro il computer. Dovreste vincere usando meno mosse. Potete specificare un pattern ideale di partenza per voi e per il computer. In questo caso il computer vi "rispecchia". Se il computer richiede più mosse rispetto a voi, allora siete un giocatore molto abile, un giocatore fortunato oppure un programmatore non eccellente. Forse state usando l'algoritmo errato!*

## CAPITOLO 6

### SPINNER

#### LE REGOLE

La posizione di LED accesa si sposta attorno al quadrato formato dai LED 1, 2, 3, 6, 9, 8, 7 e 4 in senso orario.



Scopo del gioco è di fermare la luce premendo il tasto corrispondente al LED nell'istante esatto in cui il LED viene acceso. Ogni volta che la luce rotante viene arrestata con successo, essa ricomincia a ruotare ad una velocità più elevata. Ogni volta che il giocatore non riesce ad arrestare il LED acceso in 32 giri, la luce si arresta brevemente sul LED # 4 e quindi riprende a ruotare a velocità più bassa. Il giocatore esperto sarà in grado di eseguire il gioco a una velocità di rotazione sempre più veloce, fino a raggiungere la massima velocità. A questo punto, tutti i LED sulla Scheda Giochi (i LED da 1 a 15), si accendono contemporaneamente. Si tratta di una vittoria ed inizia un nuovo gioco.

Ogni singola vittoria (arresto della luce accesa) viene indicata al giocatore da un ritardo dell'accensione del LED corrispondente al tasto premuto. Alla vittoria di una partita completa, tutti i LED della Scheda Giochi vengono accesi.

Questo gioco serve per provare i riflessi del giocatore, oppure il suo tempo di reazione. In alcuni casi la reazione del giocatore può essere troppo lenta da non riuscire a fermare la rotazione dei LED anche alla minima velocità. In questo caso il giocatore può essere autorizzato a premere due, oppure anche tre tasti consecutivi alla volta. Questo estende il tempo di risposta del giocatore. Per esempio, con questo programma, se il giocatore preme i tasti 7, 8 e 9 contemporaneamente, la luce si arresterebbe se si trovasse in una qualsiasi di queste posizioni (7, 8 oppure 9).

## L'ALGORITMO

La Fig. 6.1 riporta il diagramma di flusso. Il gioco può essere eseguito ad otto livelli di difficoltà, corrispondenti alle successive velocità del "blip" (segnale di ritorno) viaggiante con velocità crescente attorno al quadrato di LED. Un registro contatore di 8 bit viene impiegato contemporaneamente per due funzioni. (Vedere la Fig. 6.2). I 3 bit inferiori di questo registro vengono impiegati come "contatori di blip" e puntano alla posizione corrente della luce sul quadrato di LED. Tre bit selezioneranno un LED su otto. I 5 bit di estrema sinistra di questo registro sono impiegati come "contatore di ciclo" per indicare quante volte il blip attraversa il ciclo. Cinque bit consentono fino a 32 ripetizioni. I LED vengono illuminati in successione incrementando questo contatore. Ogni volta che il contatore di blip passa da 8 a 0 si introduce un riporto nel contatore del ciclo, in modo da incrementarlo automaticamente. Questi due contatori concettualmente diversi vengono allocati nel registro Y in modo da facilitare la programmazione. Naturalmente si potrebbe adottare una convenzione diversa.

Ogni volta che si accende un LED, viene esplorata la tastiera per determinare se è stato premuto il tasto corrispondente. Notare che, se il tasto era premuto prima dell'accensione del LED, esso sarà ignorato. Questo viene realizzato per mezzo di un "flag non valido". Quindi l'algoritmo controlla se era inizialmente premuto un tasto e quindi ignora qualsiasi eventuale chiusura successiva. Si ottiene una costante di ritardo moltiplicando il livello di difficoltà per quattro. Quindi, durante il ritardo in cui il LED è acceso, viene eseguito un nuovo controllo per vedere se nessun tasto era premuto all'inizio di questa routine. Se un tasto era stato premuto all'inizio, esso verrà omissso ed il programma non eseguirà alcun controllo di chiusura tasto, in quanto sarà stato posto ad uno il "flag non valido".

Ogni volta che viene premuto il tasto corretto durante il ritardo in cui il LED è acceso (ramificazione a sinistra sul diagramma di flusso nella parte centrale della Fig. 6.1), il valore del livello di difficoltà viene

decrementato (un numero inferiore del livello di difficoltà implica una velocità di rotazione più elevata). Per ogni sbaglio da parte del giocatore, il valore della difficoltà viene incrementato fino a 15, con conseguente riduzione della velocità di rotazione della luce. Una volta raggiunto il livello di difficoltà 0, se viene registrato un successo, tutti i LED sulla scheda si accenderanno per riconoscere la situazione.

## IL PROGRAMMA

### Strutture dei dati

Il programma impiega due tabelle. La tabella KYTBL memorizza i numeri dei tasti corrispondenti alla sequenza circolare di LED: 1, 2, 3, 6, 9, 8, 7, 4. Essa si trova agli indirizzi di memoria da 0B a 12. La Fig. 6.3 riporta il listing del programma.

La seconda tabella, LTABLE, contiene la struttura di bit richiesta che deve essere inviata alla porta del VIA per illuminare i LED in sequenza. Per esempio, per illuminare il LED # 1, occorre inviare la configurazione di bit "00000001" oppure 01 esadecimale. Per illuminare il LED # 2 deve essere inviata la configurazione di bit "00000010", ovvero 02 esadecimale. Analogamente per gli altri LED la struttura richiesta è rispettivamente: 04, 20, 00, 80, 40, 0B in esadecimale.

Notate che esiste un'eccezione per il LED # 9. La struttura corrispondente è 0 per la Porta 1, mentre il bit 0 della Porta 2 deve essere posto ad uno. Successivamente è necessario controllare questa situazione speciale.

### Implementazione del programma

Nella pagina 0 della memoria sono memorizzate tre variabili:

DURAT	È il ritardo tra due accensioni di LED consecutive
DIFCLT	È il "livello di difficoltà" (inverso)
DNTST	È un flag impiegato per rilevare una chiusura di tasto non consentita durante la scansione dei tasti.

Come al solito, il programma inizializza i tre registri di direzione dati richiesti: DDR1 sulla Porta A e sulla Porta B per i LED e DDR3B per la tastiera:

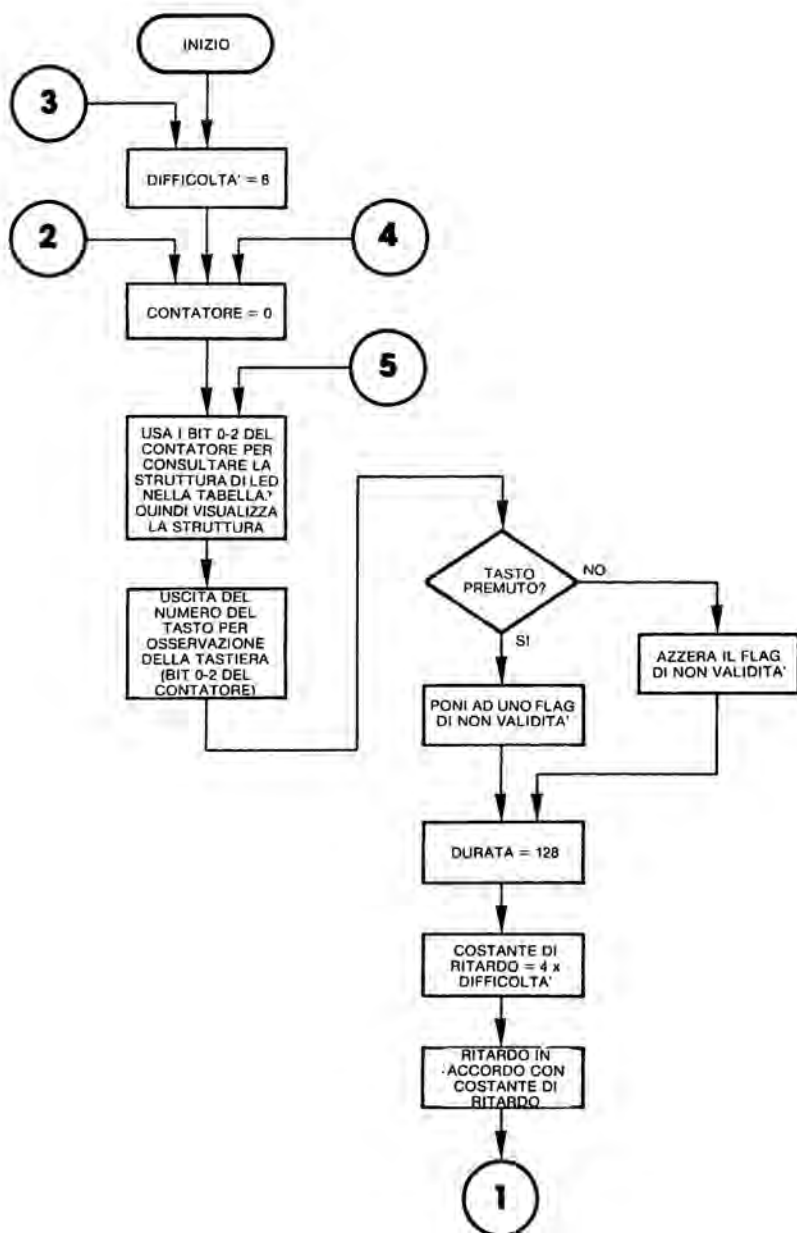


Figura 6.1: Diagramma di flusso dello Spinner

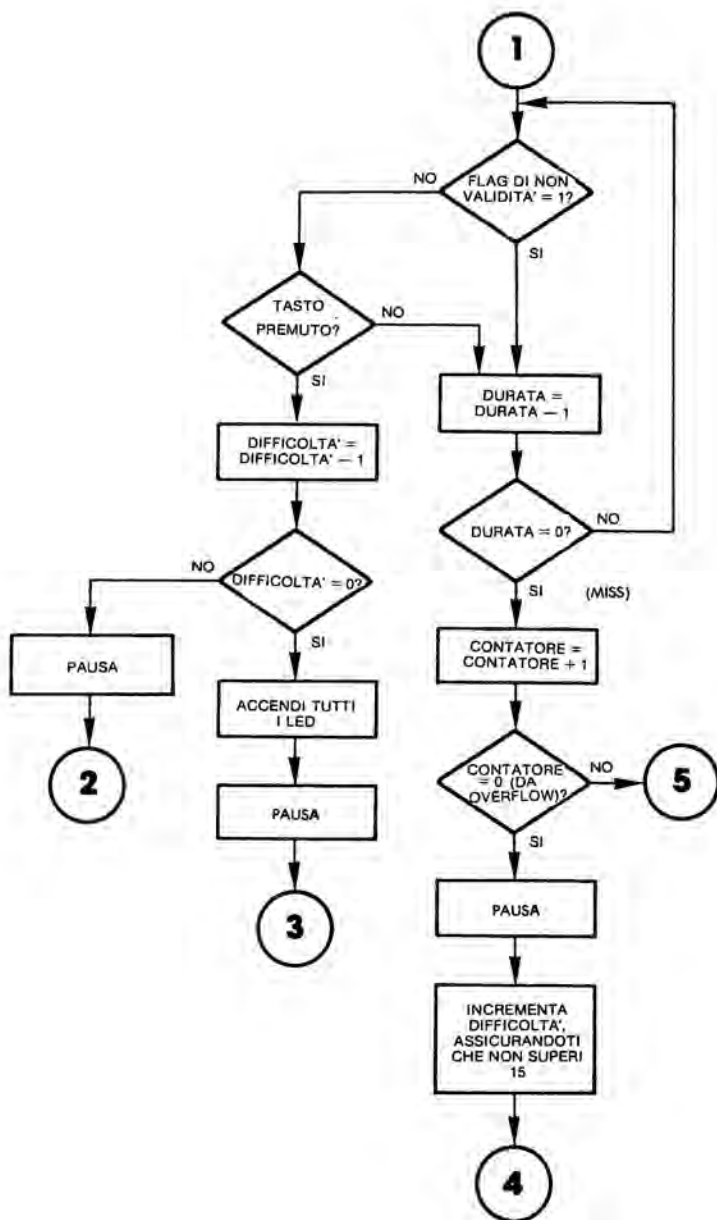


Figura 6.1: Diagramma di flusso dello Spinner (continua)

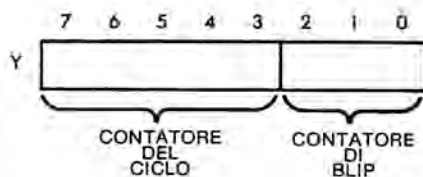


Figura 6.2: Contatore doppio

```
START      LDA #$FF
           STA DDRIA
           STA DDRIB
           STA DDR3B
```

Il livello di difficoltà viene posto ad 8, un valore intermedio;

```
LDA #8
STA DFCLT
```

La porta di strobe del tasto viene condizionata come ingresso:

```
STA DDR3A
```

Il registro Y, da impiegare come contatore il blip più ciclo generalizzato, viene posto a "0":

```
NWGME     LDY #0
```

L'indicatore di tasto premuto viene anch'esso posto a "0":

```
LOOP      LDA #0
           STA DNTST
```

Il LED # 9 viene azzerato:

```
STA PORT1B
```

Vengono estratti i tre bit inferiori del contatore. Essi contengono il contatore di blip e sono impiegati come indice nella tabella del pattern di LED:

TYA	Y contiene il contatore
AND #\$07	Estrai i 3 bit inferiori
TAX	Impiegali come indice



```

LINE  # LOC  CODE  LINE
0002  0000      ; 'SPINNER'
0003  0000      ; PROGRAMMA PER VERIFICARE IL TEMPO DI REAZIONE DEL
      ; GIOCATORE.
0004  0000      ; I LED DI UNA MATRICE 3x3 ESEGUONO UNA ROTAZIONE
0005  0000      ; E L'UTENTE DEVE PREMERE IL TASTO CORRISPONDENTE AL
0006  0000      ; LED ACCESO, SE DOPO UN CERTO NUMERO DI ROTAZIONI,
0007  0000      ; NON È STATO PREMUTO IL TASTO CORRETTO, LA ROTAZIONE
      ; DEL BLIP RALLENTA
0008  0000      ; SE È STATO PREMUTO IL TASTO CORRETTO, LA ROTAZIONE
0009  0000      ; DEL BLIP DIVENTA PIU' VELOCE SI ACCENDONO TUTTI I LED
0010  0000      ; QUANDO SI PREME CORRETTAMENTE IL TASTO ALLA
      ; MASSIMA VELOCITA'
0012  0000
0013  0000      ; I/O :
0014  0000
0015  0000      PORT1A = $A001      ; LED 1-8
0016  0000      PORT1B = $A000      ; LED 8-15
0017  0000      DDR1A = $A003
0018  0000      DDR1B = $A002
0019  0000      PORT3A = $AC01      ; INGRESSO STROBE DEL TASTO.
0020  0000      PORT3B = $AC00      ; USCITA # TASTO.
0021  0000      DDR3A = $AC03
0022  0000      DDR3B = $AC02
0023  0000
0024  0000      ; MEMORIZZAZIONE VARIABILE:
0025  0000
      ;
0026  0000      * = $ 0
0027  0000
0028  0000      DURAT *=*+1      ; DURATA DEL RITARDO TRA I
      ; MOVIMENTI.
0029  0001      DIFCLT *=*+1      ; LIVELLO DI DIFFICOLTA'.
0030  0002      DNTST *=*+1      ; IMPOSTALO A $01 SE TASTO
      ; PREMUTO ALL'INIZIO
      ; DEL RITARDO TRA I MOVIMENTI.
0031  0003
0032  0003      ;
0033  0003      ; TABELLA DELLE STRUTTURE DA INVIARE
0034  0003      ; ALLA MATRICE DI LED AD OGNI CONTEGGIO DEL CICLO.
0035  0003      ; PREDISPONI INIZIO ROTAZIONE IN SENSO ORARIO AL LED #1.
0036  0003
0037  0003      ; LTABLE .BYTE $01,$02,$04,$20,$00,$80,$40,$08
0037  0004      01
0037  0005      02
0037  0006      04
0037  0007      20
0037  0008      00
0037  0009      80
0037  000A      40
0037  000B      08
0038  000B
0039  000B      ;
0040  000B      ; TABELLA DELLE STRUTTURE DA INVIARE ALLA TASTIERA.
      ; PER VERIFICARE SE I LED SONO ACCESI AD OGNI
      ; CONTEGGIO DEL CICLO.
0041  000B
0042  000B      ; KYTBL .BYTE 1,2,3,6,9,8,7,4
0042  000C      01
0042  000D      02
0042  000E      03
0042  000F      06
0042  0010      09
0042  0011      08
0042  0012      07
0042  0012      04

```

Figura 6.3: Programma dello Spinner



0089	0252	C6 00	NOTST	DEC	DURAT	: CONTO ALLA ROVESCIA DEL
0090	0254	D0 E7		BNE	DL1	: CICLO DI RITARDO.
0091	0256	C8		INY		: RICICLA SE NON 0.
						: INCREMENTA IL CONTATORE DI
0092	0257	D0 BB		BNE	LOOP	: ROTAZIONE PRINCIPALE.
						: SE NON ESEGUITI 32 CICLI,
0093	0259	A6 01		LDX	DIFCLT	: ESEGUI ALTRO CICLO.
						: NESSUN SUCCESSO QUESTA
0094	025B					: VOLTA, VA ALLA SUCCESSIVA
0095	025B	E8		INX		: EASIER.
0096	025C	8A		TXA		: ASSICURATI CHE DIFFICOLTA'
0097	025D	C9 10		CMP	# 16	: NON SUPERI 15
0098	025D	D0 02		BNE	OK	
0099	0261	A9 0F		LDA	# 15	
0100	0263	85 01	OK	STA	DIFCLT	
0101	0265	20 80 02		JSR	WAIT	: PAUSA DI UN BIT.
0102	0268	4C 12 02		JMP	NWGME	: INIZIA NUOVO ROUND.
0103	026B	20 80 02	HIT	JSR	WAIT	: PAUSA DI UN BIT.
0104	026E	C6 01		DEC	DIFCLT	: RENDI PIU' DIFFICILE NUOVO
						: GIOCO.
0105	0270	D0 A0		BNE	NWGME	: SE DIFFICOLTA' NON 0 (PIU'
						: DIFFICILE)
0106	0272					: ESEGUI NUOVO GIOCO.
0107	0272	A9 FF		LDA	# \$FF	: IL GIOCATORE HA ESEGUITO IL
						: LIVELLO DI
0108	0274	8D 01 A0		STA	PORT1A	: MASSIMA DIFFICOLTA', ACCENDI
						: TUTTI I LED.
0109	0277	8D 00 A0		STA	PORT1B	
0110	027A	20 80 02		JSR	WAIT	: PAUSA DI UN BIT.
0111	027D	4C 00 02		JMP	START	: ESEGUI UN ALTRO GIOCO.
0112	0280					
0113	0280					: SUBROUTINE 'WAIT'
0114	0280					: RITARDO BREVE.
0115	0280					
0116	0280	A0 FF	WAIT	LDY	# \$ FF	
0117	0282	A2 FF	LP1	LDX	# \$ FF	
0118	0284	66 00	LP2	ROR	DURAT	
0119	0286	26 00		ROL	DURAT	
0120	0288	66 00		ROR	DURAT	
0121	028A	26 00		ROL	DURAT	
0122	028C	CA		DEX		
0123	028D	D0 F5		BNE	LP2	
0124	028F	88		DEY		
0125	0290	D0 F0		BNE	LP1	
0126	0292	60		RTS		
0127	0293			.END		

#### SYMBOL TABLE

#### SYMBOL VALUE

CHECK	022B	DDR1A	A003	DDR1B	A002	DDR3A	AC03
DDR3B	AC02	DELAY	0239	DIFCLT	0001	DL1	023D
DL2	0242	DNTST	0002	DURAT	0000	HIT	026B
INVALID	0235	KYTB	000B	LOOP	0214	LP1	0282
LP2	0284	LTABLE	0003	NOTST	0252	NWGME	0212
OK	0263	PORT1A	A001	PORT1B	A000	PORT3A	AC01
PORT3B	AC00	START	0200	WAIT	0280		
END OF ASSEMBLY							

Figura 6.3: Programma dello Spinner (continua)

La struttura viene ottenuta da LTABLE, impiegando un meccanismo di indirizzamento indicizzato con il registro X, e questa configurazione viene inviata in uscita sulla Porta 1A per accendere il LED appropriato:

LDA LTABLE,X	Accetta la configurazione
STA PORT1A	Impiegala per illuminare i LED

Come si è visto al paragrafo precedente, occorre eseguire un controllo esplicito per la struttura "0", la quale richiede che il bit 0 della Porta B sia posto ad uno. Questo corrisponde al LED # 9:

BNE CHECK	La struttura era =0?
LDA #1	Se no poni LED # 9
STA PORT1B	

Una volta acceso il LED corretto, la tastiera deve essere esplorata per determinare se il giocatore ha già premuto il tasto corretto. Il programma controlla soltanto il numero di tasto corrispondente al LED che deve essere acceso:

CHECK	LDA KYTBL,X	X contiene il puntatore corretto
	STA PORT3B	Seleziona il tasto corretto
	BIT PORT3A	Strobe attivo?
	BMI DELAY	Se no, salta

Se il tasto corrispondente è premuto (strobe attivo sulla Porta 3A), viene posto ad uno il flag di tasto premuto DNTST:

INVALID	LDA #01
	STA DNTST

Questa è una chiusura di tasto non consentita. Essa verrà ignorata. Viene implementato un ritardo per mantenere il LED acceso caricando un valore nella locazione di memoria DURAT. Questa locazione viene impiegata come contatore di ciclo. Essa sarà successivamente decrementata e genererà una ramificazione indietro alla locazione DL1:

DELAY	LDA #\$80
	STA DURAT

Il contatore di difficoltà, DIFCLT, viene quindi moltiplicato per quattro. Questo viene eseguito da due scorrimenti successivi a sinistra:

DL1	LDA DIFCLT
	ASL A
	ASL A
	TAX

Il risultato viene salvato nel registro indice X. Esso determinerà la lunghezza del ritardo. Al diminuire del livello di difficoltà, diminuirà la durata del ritardo.

Viene quindi implementato il ciclo di ritardo:

DL2	ROL DNTST	
	ROR DNTST	
	DEX	
	BNE DL2	Ricicla finchè conteggio =0

Il flag di tasto premuto, DNTST, viene quindi prelevato dalla memoria e controllato. Se il tasto era premuto all'inizio di questa routine, si ha la ramificazione della locazione NOTST. Altrimenti, se si verifica una chiusura, si ha la ramificazione alla locazione HIT (battuta):

LDA DNTST	
BNE NOTST	
BIT PORT3A	Controlla strobe del tasto
BPL HIT	

A NOTST prosegue l'esecuzione del ciclo di ritardo esterno; il valore di DURAT viene decrementato e si verifica una ramificazione indietro alla locazione DL1, a meno che DURAT non sia decrementata a "0". Ogni volta che il ritardo è decrementato a "0" senza che si sia riscontrata la pressione del tasto corretto, il contatore principale (il registro Y) viene incrementato di 1. Questo fa avanzare il contatore di blip (i tre bit inferiori del registro Y) al LED successivo. Comunque, se il contatore di blip stata puntando al LED # 4 (l'ultimo della nostra sequenza), il contatore del ciclo (i 5 bit superiori del registro Y) sarà automaticamente incrementato di 1 quando avanza il contatore di blip. Se il contatore del ciclo raggiunge il valore 32, il valore del registro Y dopo l'operazione di incremento sarà "0" (infatti si è verificato un overflow nel bit carry). Questa condizione viene verificata esplicitamente:

NOTST	DEC DURAT	
	BNE DL1	Ricicla se non 0
	INY	Incrementa contatore
	BNE LOOP	32 cicli?

Una volta verificatosi l'overflow del registro Y, cioè dopo l'esecuzione di 32 cicli, il valore di difficoltà viene aumentato, con conseguente riduzione della velocità di rotazione:

LDX DIFCLT	Insuccesso. Rendi più facile
INX	

Il massimo livello di difficoltà è 15 e questo viene verificato esplicitamente:

	TXA	Solo A può essere confrontato
	CMP #16	
	BNE OK	
	LDA #15	Arriva al massimo di 15
OK	STA DIFCLT	

Infine viene implementata una breve pausa:

JSR WAIT

ed inizia una nuova rotazione:

JMP NWGME

Anche in caso di successo viene implementata una pausa:

HIT JSR WAIT

quindi il gioco viene reso più difficile decrementando il conteggio di difficoltà (DIFCLT)

DEC DIFCLT

Quindi si verifica se il valore di difficoltà è "0" (rotazione più veloce possibile). Se si è raggiunto il livello "0", il giocatore ha vinto il gioco e vengono accesi tutti i LED:

BNE NWGME	Se non 0, esegui un altro gioco
LDA #\$FF	È una vittoria
STA PORT1A	Accendi i LED
STA PORT1B	

Viene implementata la solita pausa ed inizia un nuovo gioco:

JSR WAIT  
JMP START

La pausa viene ottenuta con la solita routine di ritardo chiamata "WAIT". Si tratta di una routine di tipo classico con ciclo di ritardo a due livelli annidati ed istruzioni aggiuntive di non operazione inserite all'indirizzo 0286 per renderla più lunga:

```
WAIT      LDY #$FF
LP1       LDX #$FF
LP2       ROR DURAT
          ROL DURAT
          ROR DURAT
          ROL DURAT
          DEX
          BNE LP2
          DEY
          BNE LP1
          RTS
```

## SOMMARIO

Questo programma implementa un gioco di abilità. Sono disponibili livelli multipli di difficoltà per rendere più arduo il compito del giocatore. Poiché il tempo di reazione umano è elevato, tutti i ritardi sono stati implementati come cicli di ritardo. Per motivi di efficienza, in un singolo registro è stato implementato un contatore doppio: il contatore di blip ed il contatore del ciclo.

## ESERCIZI

**Esercizio 6-1:** *Esistono diversi modi per "ingannare" questo programma. Qualsiasi tasto può essere fatto oscillare rapidamente. Inoltre è possibile premere un numero qualsiasi di tasti contemporaneamente aumentando in modo massiccio le probabilità di successo. Modificate il programma precedente in modo da prevenire queste due possibilità.*

**Esercizio 6-2:** *Variate la velocità di rotazione della luce attorno ai LED modificando la locazione di memoria appropriata. (Suggerimento: Questa locazione di memoria ha un nome indicato all'inizio del programma).*

**Esercizio 6-3:** *Aggiungete degli effetti sonori.*





# SLOT MACHINE

### LE REGOLE

Questo programma simula una slot machine del tipo che si possono trovare a Las Vegas. La rotazione delle ruote di una slot machine è simulata da tre linee verticali di luce sulle colonne di LED 1-4-7, 2-5-8 e 3-6-9. Le luci "ruotano" attorno a queste tre colonne ed eventualmente si arrestano. (Vedere Fig. 7.1). La combinazione finale di luce rappresentante il risultato del giocatore è formata dai LED 4-5-6, cioè dalla riga orizzontale centrale.

All'inizio di ciascun gioco, il giocatore ha un punteggio di otto punti. Il punteggio del giocatore viene visualizzato dal LED corrispondente sulla Scheda Giochi. All'inizio di ciascun gioco, è acceso il LED # 8, indicando un punteggio iniziale pari ad 8.

Il giocatore fa partire la slot machine premendo un tasto qualsiasi. Le luci iniziano a ruotare sulle tre righe verticali di LED. Una volta che si arrestano, la combinazione di luci sui LED 4, 5 e 6 determina il nuovo punteggio. Se nella riga centrale è acceso un solo LED o nessuno si ha una situazione perdente ed il giocatore perde un punto. Se sono accesi due LED nella riga centrale, il punteggio viene aumentato di un punto. Se sono accesi tre LED nella riga centrale, vengono aggiunti tre punti al punteggio del giocatore.

Ogni volta che si ottiene un risultato totale zero, il giocatore ha perso il gioco. Il giocatore vince se il suo punteggio arriva a 16. In questi casi viene emesso un suono dalla macchina. Mentre i LED stanno ruotando, l'altoparlante emette un crepitio, rafforzando la sensazione di movimento. Ogni volta che le luci rotanti si arrestano, l'altoparlante emette un suono, acuto se si tratta di una situazione vincente, basso se perdente. In particolare, nel caso di tre luci accese nella riga centrale (una situazione vincente), l'altoparlante emette un beep-beep-beep acuto per sottolineare il fatto che il punteggio deve essere incrementato di tre punti. Ogni

volta che si raggiunge un massimo di 16 punti il giocatore ha ottenuto un "jackpot", ossia ha vinto il gioco. A questo punto tutti i LED sulla scheda si accendono contemporaneamente e viene generato un suono di sirena (con intensità crescente). Inversamente, se si raggiunge un risultato zero, viene emesso un suono di sirena di intensità decrescente.

Notare che, diversamente dal modello di Las Vegas, questa macchina vi consente di vincere frequentemente! Buona fortuna. Comunque, come sapete, non è solo un problema di fortuna ma è una conseguenza del metodo di programmazione (come nelle macchine di Las Vegas). Trovate che l'impostazione del punteggio e le probabilità di vincita possono essere facilmente modificate attraverso il programma.

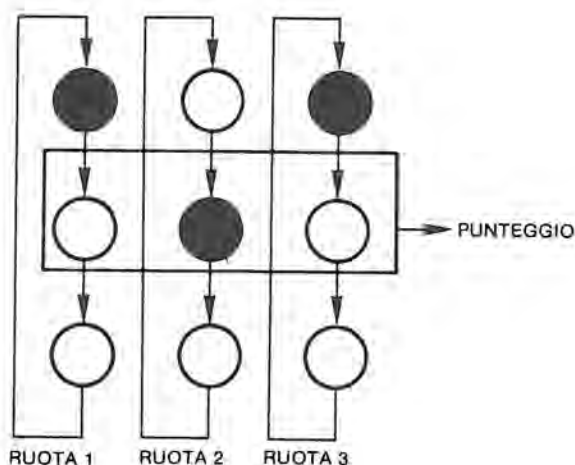


Figura 7.1: La slot machine

## UNA GIOCATA TIPICA

La Scheda Giochi visualizza inizialmente un LED illuminato nella posizione 8. A questo punto il giocatore dovrebbe selezionare e premere un tasto. Per esempio, supponiamo che venga premuto il tasto 0. Le luci cominciano a ruotare. Alla fine di questa rotazione, sono accesi i LED 4, 5 e 9. (Vedere la Fig. 7.2). Questa è una situazione vincente e verrà sommato un punto al punteggio. Viene emesso un tono acuto. Viene quindi acceso il LED # 9 per indicare la somma degli 8 punti precedenti e del punto ottenuto in questa rotazione.

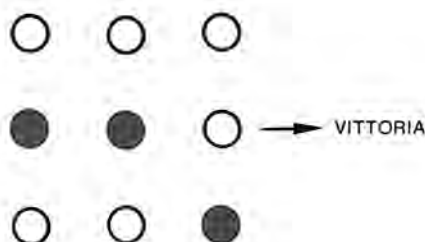


Figura 7.2: Una situazione vincente

Viene nuovamente premuto il tasto 0. Questa volta, dopo la rotazione, rimane acceso soltanto il LED 5 della riga centrale. Il punteggio ritorna ad 8. (Ricordate che il giocatore perde un punto del suo punteggio se si accende un solo LED o nessuno nella riga centrale dopo la rotazione).

Viene premuto nuovamente il tasto 0. Questa volta i LED 5 e 6 rimangono accesi ed il punteggio va a 9.

Viene premuto ancora il tasto 0; alla fine della rotazione rimane acceso il LED 4 e si accende nuovamente il LED 8.

Viene premuto il tasto 0. Si accende il LED 6, il risultato va ora a 7 e così via.

## L'ALGORITMO

La sequenza fondamentale del programma di una slot machine è riportata nel diagramma di flusso della Fig. 7.3. In primo luogo viene visualizzato il punteggio, quindi si inizia il gioco con la pressione di un tasto da parte del giocatore ed i LED vengono fatti ruotare. Successivamente vengono valutati i risultati: il punteggio viene corrispondentemente aggiornato e viene segnalata una situazione vincente o perdente.

Indichiamo le posizioni dei LED di una colonna con 0, 1, 2 dall'alto in basso. I LED sono fatti ruotare sequenzialmente illuminando le posizioni 0, 1, 2 e poi ritornando alla posizione 0. I LED continuano a ruotare in questo modo e la loro velocità di rotazione diminuisce fino ad arrestarsi. Questo effetto viene ottenuto incrementando il ritardo tra ciascuna attivazione successiva di un LED di una certa colonna. Un registro contatore viene associato ad ogni "ruota" o colonna di tre LED. I contenuti iniziali dei tre contatori delle ruote 1, 2 e 3 si ottengono da un generatore di numeri casuali. Per influenzare la casualità, il numero casuale deve essere scelto all'interno di un gruppo programmabile chiamato (LOLIM, HILIM). Il valore di questo contatore viene trasferito ad una locazione di memoria temporanea. Questa locazione viene regolar-

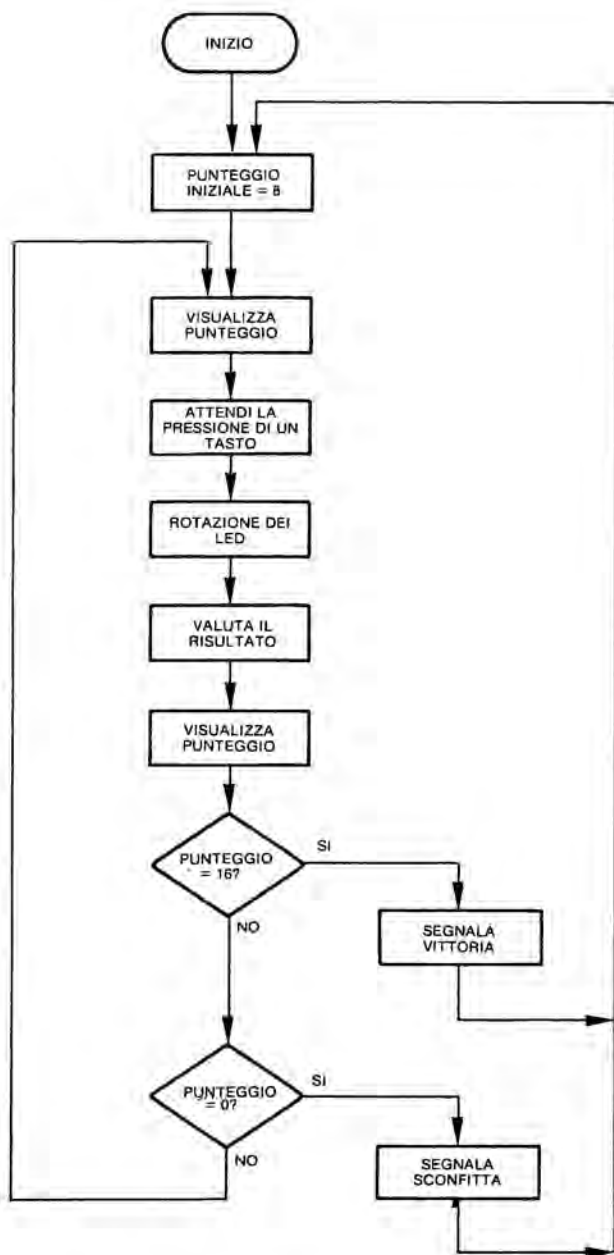


Figura 7.3: Diagramma di flusso della slot machine

mente decrementata fino a raggiungere il valore "0". Quando è stato raggiunto il valore "0", viene illuminato il successivo LED sulla "ruota". Inoltre i contenuti originari del contatore vengono incrementati di uno, con conseguente aumento del ritardo prima dell'accensione del LED successivo. Ogni volta che si ha l'overflow del contatore a 0, il processo della ruota relativa si arresta. Quindi, impiegando un aggiornamento sincrono delle locazioni di memoria temporanee, si ottiene un effetto di movimento asincrono sul "blip" (segnale di ritorno) sui LED. Quando tutti i LED si sono arrestati, è possibile valutare la situazione risultante.

La Fig. 7.4 riporta il diagramma di flusso corrispondente a questa routine DISPLAY. Ai passi 1, 2 e 3 i puntatori dei LED sono inizializzati alla riga in alto dei LED stessi (posizione 0). I tre contatori impiegati per fornire l'intervallo di temporizzazione di ciascuna ruota sono caricati con numeri provenienti dal generatore di numeri casuali. Il numero casuale viene selezionato tra i limiti imposti. Infine i tre contatori sono copiati nelle locazioni temporanee riservate per decrementare le costanti di ritardo.

Esaminiamo i passi successivi presentati in Fig. 7.4:

4. Il puntatore della ruota X viene impostato nella colonna di estrema destra:  $X = 3$ .
5. Il puntatore corrispondente della colonna corrente (colonna 3 in questo caso) viene controllato per vedere se è 0, cioè se la ruota si è fermata. Inizialmente questo valore non è 0.
- 6,7. La costante di ritardo della colonna di LED determinata dal puntatore della ruota viene decrementata, quindi si esegue il test rispetto al valore 0. Se il ritardo non è 0, si passa a sinistra di una posizione di colonna:
  16. Il puntatore di colonna X viene decrementato:  $X = X - 1$
  17. Si verifica se X è 0. In caso affermativo, si verifica una ramificazione al passo 5. Ogni volta che X raggiunge il valore 0, la stessa situazione può essersi verificata in tutte e tre le colonne. Quindi, si controllano tutti i contatori di ruota per vedere se sono 0.
  18. Se tutti i contatori sono 0, la rotazione è terminata e si esce. Se tutti i contatori non sono zero, viene implementato un ritardo e si ha una ramificazione indietro al passo 4.

Ritorniamo al passo 7:

7. Se la costante di ritardo ha raggiunto il valore 0, deve essere acceso il LED successivo in basso nella colonna.
8. Il puntatore del LED della ruota il cui numero si trova nel puntatore di ruota, viene incrementato.
9. Il puntatore di LED viene confrontato con il valore 4. Se non si è raggiunto il valore 4 si procede; altrimenti si ha il reset al

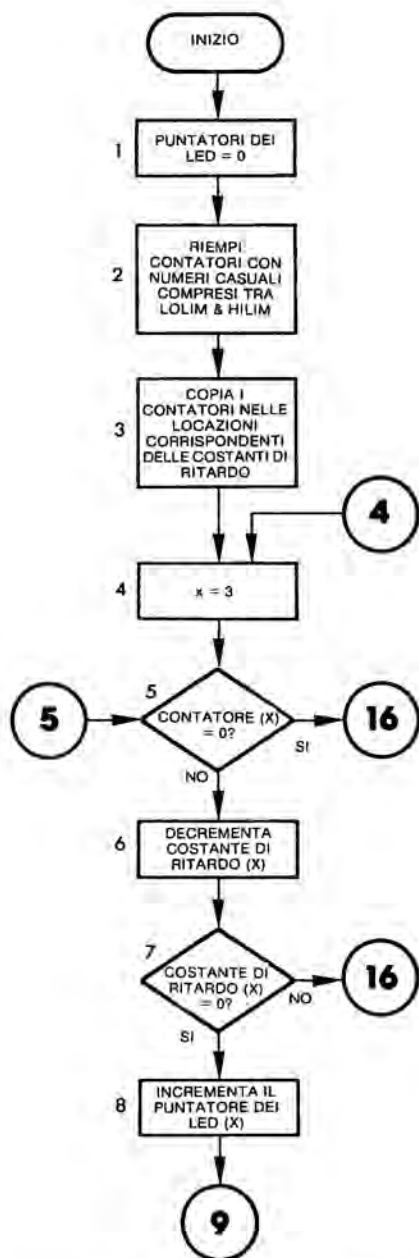


Figura 7.4: Diagramma di flusso DISPLAY

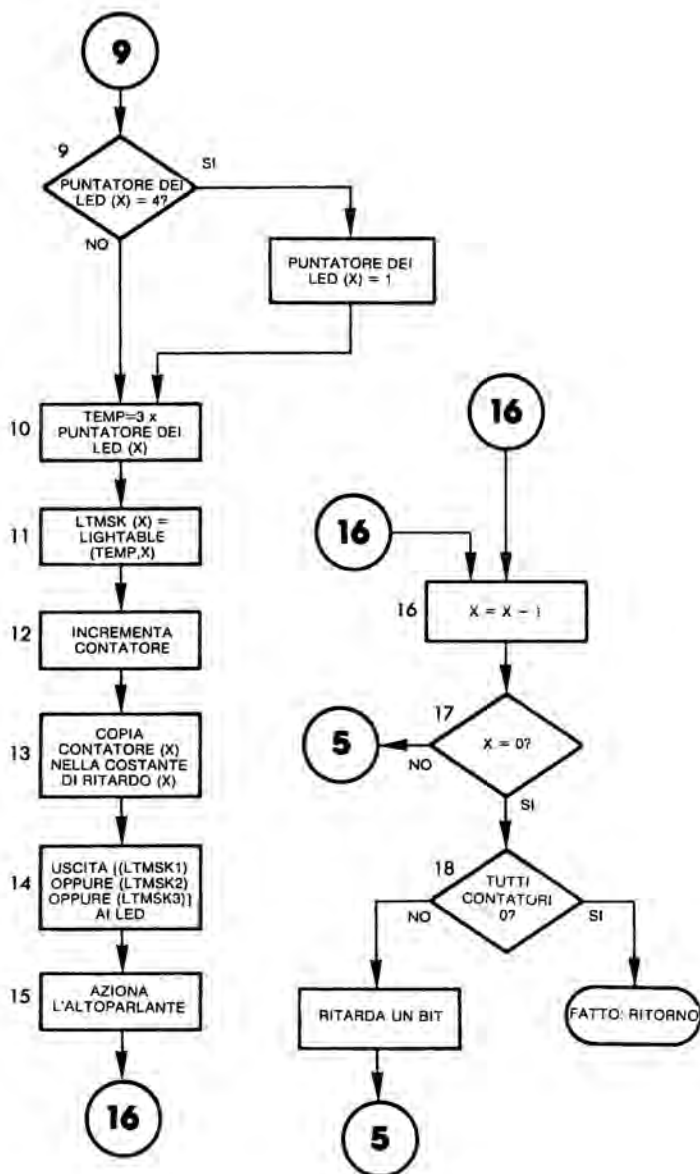


Figura 7.4: Diagramma di flusso DISPLAY (continua)

valore 1. (I LED sono contrassegnati esternamente dalle posizioni 1, 2 e 3 dall'alto al basso. Il LED successivo da accendere dopo il LED # 3 è il LED # 1).

- 10,11. I LED devono essere illuminati ed una tabella LIGHTABLE viene impiegata per ottenere la struttura corretta.
12. Viene incrementato il contatore della ruota corretta. Notate che non viene confrontato con il valore zero. Questo verrà fatto soltanto quando il programma ritorna a sinistra della ruota 1. Questo viene eseguito alla locazione 18 del diagramma di flusso, dove si confrontano i contenuti con il valore 0.
13. Il nuovo valore del contatore viene copiato nella locazione della costante di ritardo, generando un aumento del ritardo prima dell'attuazione del LED successivo.
14. Le strutture di illuminazione correnti di ciascuna colonna vengono combinate e visualizzate.
15. All'accensione in sequenza di ciascun LED si ha l'attivazione dell'altoparlante.
16. Come al solito, si passa alla colonna sinistra e si procede come prima.

Ritorniamo indietro per verificare il passo 5 del diagramma di flusso:

5. Notate che ogni qualvolta il valore contatore di una colonna è zero, il LED di quella colonna ha arrestato il movimento. Non è richiesta un'ulteriore accensione. Nel diagramma di flusso si tiene conto di ciò con la freccia a destra nel blocco di decisione a 5: la ramificazione si verifica a 16 ed il puntatore di colonna viene decrementato, e conseguentemente non viene introdotta alcuna variazione per le colonne i cui contatori erano 0.

Successivamente, l'algoritmo di valutazione deve valutare il risultato una volta che tutti i LED si sono fermati e quindi segnalare i risultati al giocatore. Consideriamo questo processo.

## Il processo di valutazione

La Fig. 7.5 riporta il diagramma di flusso dell'algoritmo EVAL. Il processo di valutazione viene inoltre illustrato in Fig. 7.6, che mostra i nove LED e le corrispondenti entità associate. Con riferimento alla Fig. 7.6, X è un puntatore di riga ed Y di colonna, o puntatore di ruota. Ad ogni riga viene associato il valore del contatore. Esso contiene il numero totale di LED illuminati in quella riga. Il valore del contatore sarà convertito in un punteggio in accordo con le regole specifiche di ciascuna riga. In questo modo abbiamo impiegato soltanto due righe ed abbiamo definito una situazione vincente come quella in cui due o tre LED sono accesi in quella riga. Comunque molte altre combinazioni sono possibili



e consentite da questo meccanismo. In seguito verranno suggeriti degli esercizi per altre configurazioni vincenti.

Il totale di tutti i punteggi di ciascuna riga sono sommati in un totale chiamato SCORE (punteggio), riportato in basso a destra in Fig. 7.6.

Facciamo ora riferimento al diagramma di flusso di Fig. 7.5. Il puntatore di colonna o di ruota Y viene settato inizialmente alla colonna di estrema destra :  $Y = 3$ .

2. I contatori temporanei sono inizializzati al valore zero.
3. All'interno della colonna corrente (3), dobbiamo osservare soltanto la riga avente un LED acceso. Questa riga viene puntata da LEDPOINTER. Il valore della riga corrispondente viene memorizzato in:  $X = \text{LEDPOINTER}(Y)$ .
4. Poichè nella riga puntata da X c'è un LED acceso, il valore del contatore di quella riga viene incrementato di uno.

Assumendo la situazione di LED mostrata nella Fig. 7.7, il valore del secondo contatore è stato posto ad 1.

5. Viene esaminata la colonna successiva:  $Y = Y - 1$ .

Se Y non è 0, si ritorna indietro a (3); altrimenti il processo di valutazione può procedere alla fase successiva.

**Esercizio 7-1:** *Impiegando il diagramma di flusso della Fig. 7.5 e l'esempio della Fig. 7.7, mostrate i valori risultanti contenuti nei contatori all'uscita del test, (6) nel diagramma di flusso della Fig. 7.5.*

Il numero effettivo di LED accesi per ogni riga deve ora essere convertito in un punteggio. Per questo scopo viene impiegata la SCORE-TABL. Qualora si cambino le regole di punteggio contenute in questa tabella, si modifica completamente il modo in cui viene eseguito il gioco.

La tabella del punteggio contiene quattro numeri lunghi un byte per ogni riga. Ogni numero corrisponde al punteggio guadagnato dal giocatore quando in quella riga sono accesi 0, 1, 2 oppure 3 LED. L'organizzazione logica della tabella del punteggio è riportata in Fig. 7.8. Gli ingressi della tabella corrispondono ai valori di punteggio che sono stati selezionati per il programma presentato all'inizio di questo capitolo. Qualsiasi combinazione di LED nella riga 1 o 3 aggiunge un punteggio 0. Qualsiasi combinazione di 2 LED nella riga 2 ha un punteggio 1 e tre LED ha un punteggio 3. Praticamente questo significa che il valore di punteggio della riga 1 viene ottenuto semplicemente usando una tecnica di accesso indicizzato con il numero di LED accesi come indice. Per la riga 2, occorre sommare uno spostamento pari a quattro per l'accesso alla tabella. Nella riga 3, occorre uno spostamento addizionale pari a quattro. Matematicamente si ha:

$$\text{SCORE} = \text{SCORETABL} [(X - 1) \times 4 + 1 + Y]$$

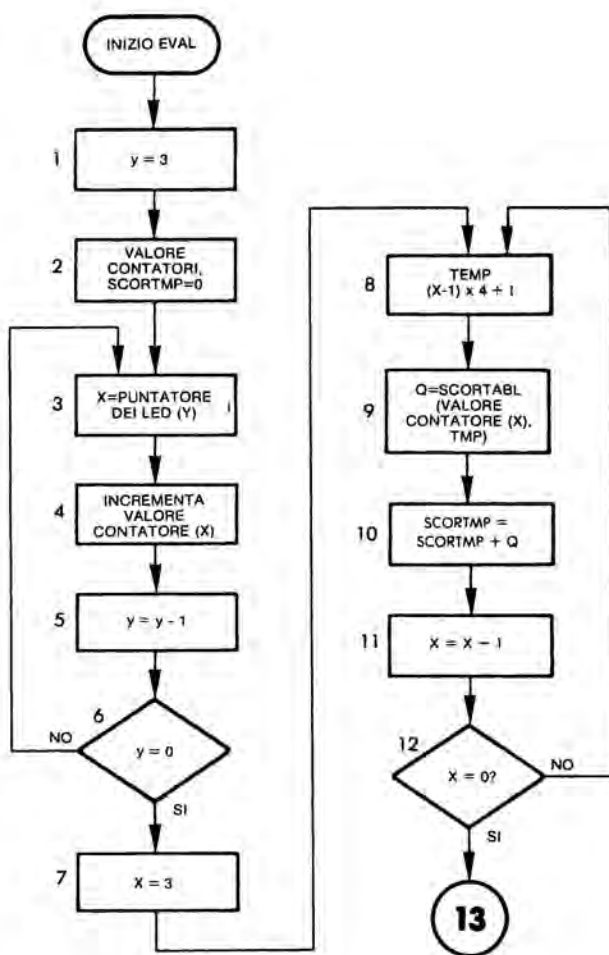


Figura 7.5: Diagramma di flusso EVAL

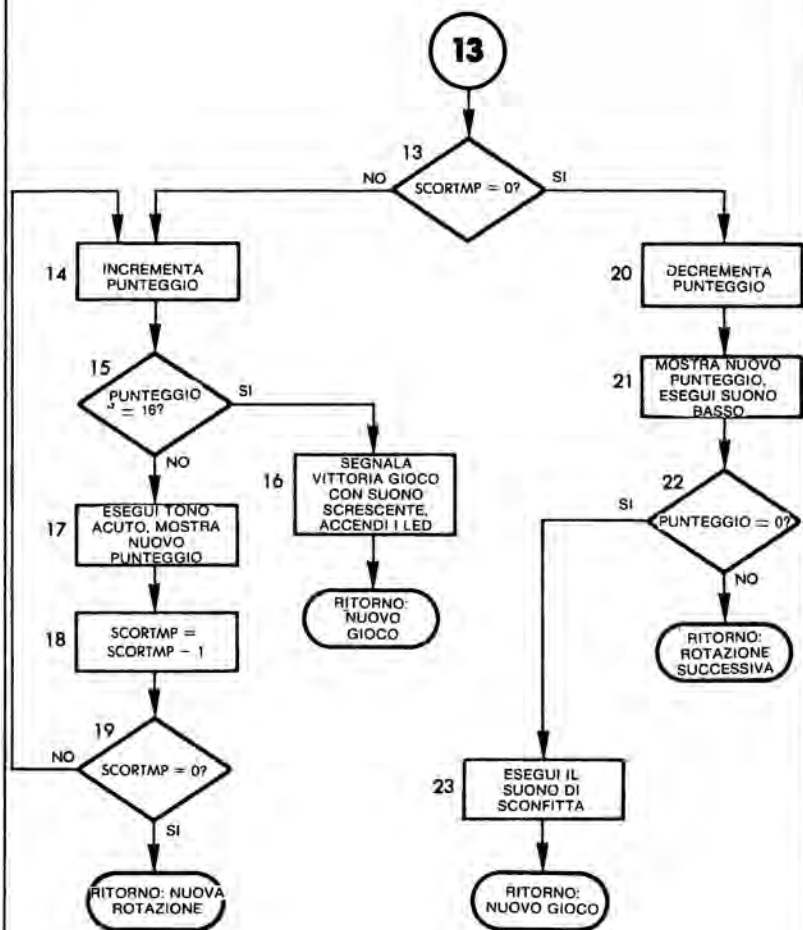


Figura 7.5: Diagramma di flusso EVAL (continua)

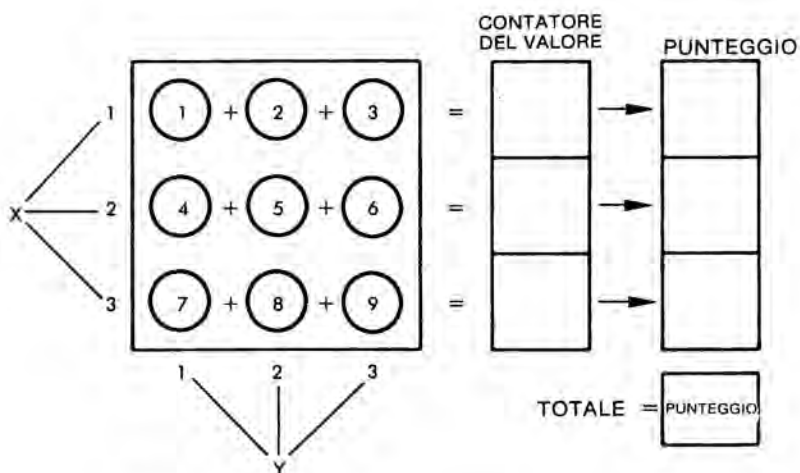


Figura 7.6: Processo di valutazione sulla scheda

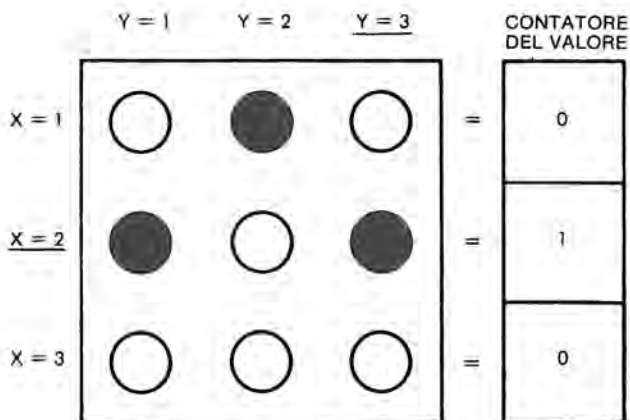


Figura 7.7: Un esempio di valutazione

dove X è il numero di riga ed Y è il numero di LED accesi di quella riga. Poiché questa tecnica consente di generare un punteggio per ciascuna delle tre righe, il programma deve eseguire il test del valore del contatore in ogni riga in modo da ottenere il punteggio totale.

Questo viene eseguito dai passi 7 ed 8: il puntatore di riga viene inizializzato a 3 e viene impostato il puntatore dello spostamento per la

0	1	2	3	NUMERO DI LED ACCESI
0	0	0	0	RIGA 1
0	0	1	3	RIGA 2
0	0	0	0	RIGA 3

Figura 7.8: La tabella del punteggio

tabella del punteggio:

$$\text{TEMP} = (X - 1) \times 4 + 1$$

9. Successivamente, si ottiene dalla tabella il valore del punteggio:

$$Q = \text{SCORTABL}(\text{valore del contatore } (X), \text{TEMP})$$

Il valore di quel punteggio di riga viene ottenuto accedendo alla tabella del punteggio indicizzata dal numero di LED accesi, contenuti nel valore del contatore di quella riga, più uno spostamento uguale TEMP. Il punteggio intermedio viene ottenuto sommando questo punteggio parziale al valore precedente:

10.  $\text{SCORTMP} = \text{SCORTMP} + Q$
11. Infine, il numero di riga viene decrementato ed il processo viene ripetuto finché X raggiunge il valore 0.
12. Ogni volta che X raggiunge il valore 0, il punteggio di questa rotazione è stato calcolato e memorizzato nella locazione SCORTMP.
13. A questo punto, il punteggio precedentemente calcolato (SCORTMP) viene esaminato dal programma ed esistono due possibilità: se SCORTMP è 0, si ha una ramificazione a 20, dove viene decrementato il punteggio del gioco. Se SCORTMP non è 0, il punteggio del gioco verrà incrementato dal punteggio di questa rotazione - SCORTMP. Consideriamo quest'ultimo caso.
14. Il punteggio totale del gioco viene incrementato di uno.
15. Il punteggio viene quindi confrontato con il valore massimo pari a 16.

16. Se al passo 15 si è raggiunto il punteggio massimo (16), viene generato uno speciale segnale visivo e sonoro per premiare il giocatore. Quindi occorre iniziare un nuovo gioco.
17. Se al passo 15 non è stato raggiunto il punteggio 16, viene mostrato al giocatore il punteggio aggiornato del gioco, seguito da un suono acuto.
18. Si decrementa la quantità di cui deve essere aumentato il punteggio del gioco, SCORTMP.
19. Se SCORTMP non è zero, occorre sommare più punti al punteggio del gioco e si ha una ramificazione a 14. Altrimenti si può passare alla rotazione successiva.

Seguiamo ora l'altra alternativa dalla posizione tredici sul diagramma di flusso, dove è stato eseguito il test del punteggio totale:

20. Il punteggio di questa rotazione è zero, quindi il punteggio del gioco viene decrementato.
21. Il punteggio del gioco viene visualizzato al giocatore assieme ad un suono grave.
22. Si esegue il test del nuovo punteggio rispetto al valore minimo 0. Se è stato raggiunto questo valore minimo, il giocatore ha perso. Altrimenti il giocatore può continuare a giocare.
23. Per indicare la perdita e la fine del gioco viene generato un suono del tipo a sirena discendente.

## IL PROGRAMMA

### Strutture dei dati

Questo programma impiega due tabelle: 1) la tabella del punteggio viene impiegata per calcolare un punteggio del numero di LED accesi in ogni riga - questa è già stata descritta; 2) la tabella LTABLE viene impiegata per generare il codice appropriato sulla porta di I/O per accendere il LED specificato. Ciascun ingresso all'interno di questa tabella contiene una configurazione per l'esecuzione dell'OR nel registro di I/O per accendere il LED specificato.

Verticalmente, in memoria, gli ingressi della tabella corrispondono alla prima, alla seconda ed alla terza colonna di LED. Osservando il programma alle linee 39, 40 e 41, le righe di digit corrispondono rispettivamente alle colonne dei LED. Per esempio, il terzo ingresso della tabella, cioè 64 decimale, oppure 40 esadecimale (all'indirizzo 001C) corrisponde al terzo LED della prima colonna sulla Scheda Giochi, ovvero al LED 7.

LINE #	LOC	CODE	LINE
0002	0000		: PROGRAMMA SIMULATORE DELLA SLOT MACHINE.
0003	0000		: PREMERE UN TASTO QUALSIASI PER INIZIARE LA
			: ROTAZIONE.
0004	0000		: IL PUNTEGGIO VIENE DETERMINATO DALLA MATRICE DI
			: "SCORTB"
0005	0000		: PUNTEGGIO INIZIALE 8 PUNTI, UN PUNTO DI PENALITA'
0006	0000		: AD OGNI ROTAZIONE CON RISULTATO NEGATIVO.
0007	0000		:
0008	0000		: * = \$0
0009	0000	TEMP	*=*+1 : MEMORIZZAZIONE
			: TEMPORANEA.
0010	0001	SCORTP	*=*+1 : MEMORIZZAZIONE DEL
			: PUNTEGGIO TEMPORANEO.
0011	0002	SCORE	*=*+1 : PUNTEGGIO.
0012	0003	DUR	*=*+1 : DURATA DEI TONI.
0013	0004	FREQ	*=*+1 : FREQUENZA DEI TONI.
0014	0005	SPEEDS	*=*+3 : VELOCITA' DI ROTAZIONE DEI
			: LED
0015	0008		: NELLE COLONNE
0016	0008	INDX	*=*+3 : CONTATORI DI RITARDO PER
			: LE ROTAZIONI DEI LED.
0017	000B	INCR	*=*+3 : PUNTATORI ALLE POSIZIONI
			: DEI LED
0018	000E		: IMPIEGATI PER PRELEVARE LE STRUTTURE
			: DALLE TABELLE.
0019	000E	LTMSK	*=*+3 : STRUTTURE PER
			: ACCENDERE I LED
0020	0011	VALUES	*=*+3 : NUMERO DI LED ACCESI PER
			: OGNI RIGA.
0021	0014	RND	*=*+6 : SCRATCHPAD PER IL GEN. DI
			: NUM. CASUALI.
0022	001A		:
0023	001A		: I/O
0024	001A		:
0025	001A	PORT1A	= \$A001 : REGISTRO DI I/O DELLA
			: PORTA A DEL VIA # 1 (LED).
0026	001A	DDR1A	= \$A003 : REGISTRO DI DIREZIONE
			: DATI DELLA PORTA A VIA # 1.
0027	001A	PORT1B	= \$A000 : REGISTRO DI I/O DELLA
			: PORTA B VIA # 1.
0028	001A	DDR1B	= \$A002 : REGISTRO DI DIREZIONE
			: DATI DELLA PORTA B VIA # 1.
0029	001A	PORT3B	= \$AC00 : REGISTRO DI I/O DELLA
			: PORTA B VIA # 3 (ALTOP)
0030	001A	DDR3B	= \$AC02 : REGISTRO DI DIREZIONE
			: DATI DELLA PORTA B VIA # 3.
0031	001A	T1CL	= \$A004
0032	001A		:
0033	001A		: MATRICI
0034	001A		:
0035	001A		: MATRICE DELLE STRUTTURE DI ACCENSIONE DEI LED.
0036	001A		: RIGHE DELLE MATRICI CORRISPONDENTI ALLE
			: COLONNE DI LED. MATRICE, DALLE COLONNE ALLE
			: RIGHE, PER ESEMPIO, IL
0037	001A		: TERZO BYTE DELLA RIGA UNO ACCENDERA' IL LED 7.
0038	001A		: LTABLE .BYTE 1,8,64
0039	001A	01	
0039	001B	08	
0039	001C	40	
0040	001D	02	.BYTE 2,16,128
0040	001E	10	
0040	001F	80	

Figura 7.9: Programma della slot machine

```

0041 0020 04          .BYTE 4,32,0
0041 0021 20
0041 0022 00
0042 0023          ; MATRICE DEI PUNTEGGI RICEVUTI PER DETERMINATE
0043 0023          ; STRUTTURE DI LED ACCESI.
0044 0023          ; LE RIGHE CORRISPONDONO A QUELLE DELLA MATRICE
          ; DI LED.
0045 0023          ; LE COLONNE CORRISPONDONO AL NUMERO DI LED
0046 0023          ; ACCESI IN QUELLA RIGA.
0047 0023          ; CIOÈ 3 LED NELLA RIGA CENTRALE CORRISPONDONO A
          ; TRE PUNTI.
          SCORTB .BYTE 0,0,0,0

0048 0023 00
0048 0024 00
0048 0025 00
0048 0026 00
0049 0027 00          .BYTE 0,0,1,3
0049 0028 00
0049 0029 01
0049 002A 03
          .BYTE 0,0,0,0
0050 002B 00
0050 002C 00
0050 002D 00
0050 002E 00
0051 002F
0052 002F          ; ***** PROGRAMMA PRINCIPALE *****
0053 002F
0054 002          FGETKEY = $100
0055 002F          * = $200
          LDA #$FF          ; SET UP PORTS.
0056 0200 A9 FF          STA DDR1A
0057 0202 8D 03 A0          STA DDR1B
0058 0205 8D 02 A0          STA DDR3B
0059 0208 8D 02 AC          LDA T1CL          ; ACCETTA SEME PER GEN. NUM.
0060 020B AD 04 A0          ; CASUALI.

0061 020E 85 15          STA RND+1
0062 0210 A9 08          LDA #8          ; IL PUNTEGGIO INIZIALE È OTTO.
0063 0212 85 02          STA SCORE
0064 0214 A8          TAY          ; VISUALIZZA IL PUNTEGGIO
          ; INIZIALE.

0065 0215 20 3D 03          JSR LIGHT
0066 0218 20 00 01 KEY          JSR GETKEY          ; QUALSIASI TASTO PREMUTO
          ; INIZIA IL PROGRAMMA
0067 021B 20 27 02          JSR DISPLY          ; ROTAZIONE DELLE RUOTE
0068 021E 20 A7 02          JSR EVAL          ; CONTROLLO E VISUALIZZAZIONE
          ; PUNTEGGIO

0069 0221 A5 02          LDA SCORE
0070 0223 D0 F3          BNE KEY          ; SE PUNTEGGIO < 0, PASSA AL
          ; GIOCO SUCCESSIVO.
0071 0225 F0 E9          BEQ START          ; SE PUNTEGGIO = 0, RESTART.
0072 0227
0073 0227          ; SUBROUTINE PER LA VISUALIZZAZIONE DEL LED IN MODO
          ; ROTANTE,
0074 0227          ; TROVA LA COMBINAZIONE DA IMPIEGARE PER
          ; DETERMINARE IL PUNTEGGIO

0075 0227
0076 0227          LOLIM = 90

```

Figura 7.9: Programma della slot machine (continua)



0077	0227			HILIM	= 135	
0078	0227			SPDPRM	= 80	
0079	0227	A9 00		DISPL	LDA # 0	: RESET DEI PUNTATORI.
0080	0229	85 08			STA INCR	
0081	022B	85 0C			STA INCR + 1	
0082	022D	85 0D			STA INCR + 2	
0083	022F	A0 02		LDRND	LDY # 2	: IMPOSTA L'INDICE PER TRE
						: ITERAZIONI.
0084	0231	20 80 03		GETRND	JSR RANDOM	: ACCETTA IL # CASUALE.
0085	0234	C9 87			CMP # HILIM	: TROPPO GRANDE?
0086	0236	B0 F9			BCS GETRND	: SE SI, ACCETTANE UN ALTRO.
0087	0238	C9 5A			CMP # LOLIM	: TROPPO PICCOLO?
0088	023A	90 F5			BCC GETRND	: SE SI, ACCETTANE UN ALTRO.
0089	023C	99 08 00			STA INDX, Y	: SALVALO NEGLI INDICI DEL
						: CICLO E
0090	023F	99 05 00			STA SPEEDS, Y	: NEI CONTATORI DI VELOCITA'
						: DEL CICLO.
0091	0242	88		DEY		
0092	0243	10 EC		BPL	GETRND	: ACCETTA IL SUCCESSIVO #
						: CASUALE.
0093	0245	A2 02		UPDATE	LDX # 2	: IMPOSTA L'INDICE PER TRE
						: ITERAZIONI.
0094	0247	B4 05		UPDTLP	LDY SPEEDS, X	: VELOCITA' (X) È = 0?
0095	0249	F0 44			BEQ NXTUPD	: SE SI, ESEGUI L'AGGIORNAMENTO
						: SUCCESSIVO.
0096	024B	D6 08			DEC INDX, X	: DECREMENTA L'INDICE DEL
						: CICLO (X)
0097	024D	D0 40			BNE NXTUPD	: SE INDICE DEL CICLO (X) < 0,
0098	024F				: ESEGUI	: L'AGGIORNAMENTO SUCCESSIVO.
0099	024F	B4 0B			LDY INCR, X	: INCREMENTA PUNTATORE (X).
0100	0251	C8			INY	
0101	0252	C0 03			CPY # 3	: PUNTATORE = 3?
0102	0254	D0 02			BNE NORST	: SE NO SALTA....
0103	0256	A0 00			LDY # 0	: ...RESET A 0 DEL PUNTATORE.
0104	0258	94 0B		NORST	STY INCR, X	: RIPRISTINA PUNTATORE (X).
0105	025A	86 00			STX TEMP	: MOLTIPLICA X PER 3 PER ACCESSO
						: ALLA MATRICE.
0106	025C	8A			TXA	
0107	025D	0A			ASL A	
0108	025E	18			CLC	
0109	025F	65 00			ACD TEMP	
0110	0261	75 0B			ADC INCR, X	: SOMMA # DI COLONNA A PTR (X)
						: PER # RIGA.
0111	0263	A8			TAY	: TRASFERISCI AD Y PER
						: INDICIZZAZIONE.
0112	0264	B9 1A 00			LDA LTABLE, Y	: ACCETTA LA STRUTTURA DI LED.
0113	0267	95 0E			STA LTMSK, X	: MEMORIZZA NELLA MASCHERA DI
						: ACCENSIONE (X).
0114	0269	B4 05		SPDUPD	LDY SPEEDS, X	: INCREMENTA VELOCITA' (X).
0115	026B	C8			INY	
0116	026C	94 05			STY SPEEDS, X	: RIMEMORIZZA.
0117	026E	94 08			STY INDX, X	: RESET INDICE DEL CICLO (X).
0118	0270	A9 00		LEDUPD	LDA # 0	: AGGIORNA LE ACCENSIONI.
0119	0272	8D 00 A0			STA PORT1B	: RESET LED #9
0120	0275	A5 10			LDA LTMSK + 2	: COMBINA LE STRUTTURE PER
						: L'USCITA.
0121	0277	D0 07			BNE OFFLD9	: SE MASCHERA #3 < 0,
						: LED 9 SPENTO.
0122	0279	A9 01			LDA # 01	: ACCENDI IL LED 9.
0123	027B	8D 00 A0			STA PORT1B	
0124	027E	A9 00			LDA # 0	: RESET DI A COSICCHÈ LA
						: STRUTTURA NON
						: SIA NEGATIVA.

Figura 7.9: Programma della slot machine (continua)

0125	0280	05 0E	OFFLD9	ORA LTMSK	; COMBINA IL RESTO DELLE ; STRUTTURE.
0126	0282	05 0F		ORA LTMSK + 1	
0127	0284	8D 01 A0		STA PORT1A	; ACCENDI I LED.
0128	0287	AD 00 AC		LDA PORT3B	; COMMUTA L'ALTOPARLANTE.
0129	028A	49 FF		EOR # \$ FF	
0130	028C	8D 00 AC		STA PORT3B	
0131	028F	CA	NXTUPD	DEX	; DECREMENTA X PER ; L'AGGIORNAMENTO SUCCESSIVO.
0132	0290	10 B5		BPL UPDTLP	SE X = 0, ESEGUI ; AGGIORNAMENTO SUCCESSIVO.
0133	0292	A0 50		LDY # SPDPRM	; RITARDO DI UN BIT PER ; RALLENTARE
0134	0294	88	WAIT	DEY	; IL LAMPEGGIO DEI LED.
0135	0295	D0 FD		BNE WAIT	
0136	0297	A5 05		LDA SPEEDS	; CONTROLLA SE TUTTE LE ; COLONNE DEI LED
0137	0299				; SONO FERME.
0138	0299	05 06		ORA SPEEDS+1	
0139	029B	05 07		ORA SPEEDS+2	
0140	029D	D0 A6		BNE UPDATE	; SE NO, ESEGUI LA SEQUENZA ; SUCCESSIVA
0141	029F				; OF UPDATES.
0142	029F	A9 FF		LDA # \$ FF	
0143	02A1	85 03		STA DUR	; RITARDO PER MOSTRARE LA ; STRUTTURA ALL'UTENTE.
0144	02A3	20 30 03		JSR DELAY	
0145	02A6	60		RTS	; TUTTI I LED FERMI, FATTO.
0146	02A7				
0147	02A7				; SUBROUTINE PER VALUTARE IL PRODOTTO DELLA ; ROTAZIONE E ; PER VISUALIZZARE MEDIANTE SUONI UNA VITTORIA, ; VITTORIA+FINE GIOCO E SCONFITTA+FINE GIOCO.
0148	02A7				
0149	02A7				
0150	02A7				
0151	02A7			HITONE = \$ 20	
0152	02A7			LOTONE = \$ F0	
0153	02A7	A9 00	EVAL	LDA # 0	; RESET DELLE VARIABILI.
0154	02A9	85 11		STA VALUES	
0155	02AB	85 12		STA VALUES+1	
0156	02AD	85 13		STA VALUES+2	
0157	02AF	85 01		STA SCORLP	
0158	02B1	A0 02		LDY # 2	; IMPOSTA L'INDICE Y PER 3 ; ITERAZIONI
0159	02B3				; PER IL CONTEGGIO DEL # DI LED ; IN OGNI RIGA.
0160	02B3	B6 0B	CNTLP	LDX INCR,Y	; CONTROLLA PUNTATORE (Y), ; SOMMANDO
0161	02B5	F6 11		INC VALUES,X	; IL # DI LED ACCESI IN OGNI RIGA.
0162	02B7	88		DEY	
0163	02B8	10 F9		BPL CNTLP	; RICICLA SE NON FATTO.
0164	02BA	A2 02		LDX # 2 IMPOSTA	; L'INDICE X PER TRE ITERAZIONI.
0165	02BC				; DEL CICLO PER TROVARE IL ; PUNTEGGIO.
0166	02BC	8A	SCORLP	TXA	; MOLTIPLICA L'INDICE PER 4 PER ; L'ACCESSO ; ALLA RIGA DELLA MATRICE.
0167	02BD				
0168	02BD	0A		ASL A	
0169	02BE	0A		ASL A	
0170	02BF	18		CLC	; SOMMA IL NUMERO DI LED ; ACCESI NELLA RIGA FINO ... ; ... AD ARRIVARE ALL'INDIRIZZO DI ; COLONNA NELLA MATRICE.
0171	02C0	75 11		ADC VALUES,X	

Figura 7.9: Programma della slot machine (continua)

0172	02C2	A8		TAY		: USALO COME INDICE
0173	02C3	B9 23 00		LDA SCORTB,Y		: ACCETTA IL PUNTEGGIO DI QUESTA ROTAZIONE.
0174	02C6	18		CLC		
0175	02C7	65 01		ADC SCORTP		: SOMMA IL PUNTEGGIO PRECEDENTE
0176	02C9					: ACCUMULATO IN QUESTO CICLO.
0177	02C9	85 01		STA SCORTP		: RIMEMORIZZA
0178	02CB	CA		DEX		
0179	02CC	10 EE		BPL SCORLP		: RICICLA SE NON FATTO
0180	02CE	A9 60		LDA # \$60 IMPOSTA		LA DURATA DEI TONI.
0181	02D0	85 03		STA DUR		
0182	02D2	A5 01		LDA SCORTP		: ACCETTA IL PUNTEGGIO DI QUESTA RITAZIONE.
0183	02D4	F0 34		BEQ LOSE		: SE PUNTEGGIO È 0, PERDI UN PUNTO.
0184	02D6	E6 02	WIN	INC SCORE		: AUMENTA IL PUNTEGGIO GLOBALE DI UNO.
0185	02D8	A4 02		LDY SCORE		: ACCETTA PUNTEGGIO
0186	02DA	C0 10		CPY # 16		: VINCI CON 16 PUNTI?
0187	02DC	F0 10		BEQ WINEND		: SÌ: VITTORIA+FINE GIOCO.
0188	02DE	20 3D 03		JSR LIGHT		: VISUALIZZA PUNTEGGIO.
0189	02E1	A9 20		LDA # HITONE		: ESEGUI UN BEEP ACUTO.
0190	02E3	20 64 03		JSR TONE		
0191	02E6	20 30 03		JSR DELAY		: RITARDO BREVE
0192	02E9	C6 01		DEC SCORTP		: DECREMENTA DI UNO IL PUNTEGGIO DA SOMMARE AL ...
0193	02EB					: PUNTEGGIO GLOBALE.
0194	02EB	D0 E9		BNE WIN		: RICICLA SE TRASFERIMENTO DEL PUNTEGGIO NON COMPLETO.
0195	02ED	60		RTS		: FATTO, RITORNA AL PROGRAMMA PRINCIPALE.
0196	02EE	A9 FF	WINEND	LDA # \$ FF		: ACCENDI TUTTI I LED PER SEGNALARE UNA VINCITA.
0197	02F0	8D 01 A0		STA PORT1A		
0198	02F3	8D 00 A0		STA PORT1B		
0199	02F6	85 00		STA TEMP		: IMPOSTA IL PARAMETRO DI FREQ PER TONO CRESCENTE.
0200	02F8	A9 00		LDA # 0		
0201	02FA	85 02		STA SCORE		: AZZERAZIONE PER FLAG RESTART.
0202	02FC	A9 04		LDA # 4		
0203	02FE	85 03		STA DUR		: DURATA BREVE PER I SINGOLI BEEP DEL TONO.
0204	0300					: ACCETTA LA FREQUENZA ...
0205	0300	A5 00	RISE	LDA TEMP		: ... DEL BEEP.
0206	0302	20 64 03		JSR TONE		: IL BEEP SUCCESSIVO SARA' PIU' ACUTO.
0207	0305	C6 00		DEC TEMP		: ESEGUI IL BEEP SUCCESSIVO SE NON FATTO.
0208	0307	D0 F7		BNE RISE		: RITORNO PER IL RESTART.
0209	0309	60		RTS		: SE ROTAZIONE NEGATIVA, PUNTEGGIO = PUNTEGGIO -1.
0210	030A	C6 02	LOSE	DEC SCORE		: VISUALIZZA PUNTEGGIO
0211	030C	A4 02		LDY SCORE		
0212	030E	20 3D 03		JSR LIGHT		
0213	0311	A9 F0		LDA # LOTONE		: ESEGUI IL SUONO BASSO DI PERDITA.
0214	0313	20 64 03		JSR TONE		
0215	0316	A4 02		LDY SCORE		: ACCETTA IL PUNTEGGIO PER VEDERE ...

Figura 7.9: Programma della slot machine (continua)

0216	0318	F0 01	BEQ	LOSEND	SE SI È FUORI GIOCO.
0217	031A	60	RTS		SE NO, RITORNO PER LA
					ROTAZIONE SUCCESSIVA.
0218	031B	A9 00	LOSEND	LDA # 0	IMPOSTA TEMP COME
0219	031D	85 00		STA TEMP	PARAMETRO DI FREQ
0220	031F	8D 01 A0		STA PORT1A	AZZERA LED # 1.
0221	0322	A9 04		LDA # 4	
0222	0324	85 03		STA DUR	
0223	0326	A5 00	FALL	LDA TEMP	
0224	0328	20 64 03		JSR TONE	ESEGUI IL BEEP.
0225	032B	E6 00		INC TEMP	IL SUONO SUCCESSIVO SARA'
					PIU' BASSO.
0226	032D	D0 F7	BNE	FALL	
0227	032F	60	RTS		RITORNO PER RESTART
0228	0330				
0229	0330				SUBROUTINE DI RITARDO DI LUNGHEZZA VARIABILE.
0230	0330				LUNGHEZZA RITARDO = (2046*[CONTENUTI DI DUR] + 10)
					PROSEC.
0231	0330				
0232	0330	A4 03	DELAY	LDY DUR	ACCETTA LUNGHEZZA DEL
					RITARDO.
0233	0332	A2 FF	DL1	LDX # \$ FF	IMPOSTA CNTR PER UN CICLO
					INTERNO DI 2040 MICROSEC.
0234	0334	D0 00	DL2	BNE * + 2	CONSUMA TEMPO.
0235	0336	CA		DEX	DECREMENTA CONTATORE DEL
					CICLO INTERNO.
0236	0337	D0 FB		BNE DL2	ESEGUI TUTTO IL CICLO INTERNO
					DECREMENTA CONTATORE DEL
0237	0339	88		DEY	CICLO ESTERNO.
0238	033A	D0 F6		BNE DL1	RICICLA SE NON FATTO.
0239	033C	60		RTS	RITORNO.
0240	033D				
0241	033D				SUBROUTINE PER ACCENDERE IL LED CORRISPONDENTE
0242	033D				AI CONTENUTI DEL REGISTRO Y ALL'ATTO DELLA CHIAMATA.
0243	033D				
0244	033D	A9 00	LIGHT	LDA # 0	AZZERA IL REG. A PER IL BIT DI
					SCORRIMENTO.
0245	033F	85 00		STA TEMP	AZZERA L'OVERFLOW FLAG.
0246	0341	8D 01 A0		STA PORT1A	AZZERA I LED BASSI.
0247	0344	8D 00 A0		STA PORT1B	AZZERA I LED ALTI.
0248	0347	C0 0F		CPY #15	CODICE PER UN BIT NON
					CONNESSO?
0249	0349	F0 01	BEQ	*+ 3	SE SI, NESSUNA VARIAZIONE.
0250	034B	88		DEY	DECREMENTA PER IL
					CONFRONTO.
0251	034C	38		SEC	PONI BIT PER FARLO SCORRERE
					IN ALTO.
0252	034D	2A	LTSHFT	ROL A	SCORRIMENTO DI BIT A SINISTRA.
0253	034E	90 05		BCC LTCC	SE CARRY È 1, SI È VERIFICATO
0254	0350				OVERFLOW NEL BYTE ALTO.
0255	0350	A2 FF		LDX # \$ FF	PONI A 1 OVERFLOW FLAG.
0256	0352	86 00		STX TEMP	
0257	0354	2A		ROL A	MUOVI IL BIT DAL CARRY.
0258	0355	88	LTCC	DEY	UN BIT IN MENO DA FAR
					SCORRERE.
0259	0356	10 F5	BPL	LTSHFT	ALTRO SCORRIMENTO SE NON
					FATTO.
0260	0358	A6 00		LDX TEMP	ACCETTA L'OVERFLOW FLAG.
0261	035A	D0 04		BNE HIBYTE	SE FLAG < 0, OVERFLOW: A

Figura 7.9: Programma della slot machine (continua)

```

0262 035C                                     : CONTIENE
0263 035C 8D 01 A0 LOBYTE STA PORT1A         : IL BYTE DI ORDINE ELEVATO.
                                           : MEMORIZZA A NEI LED DI BASSO
                                           : ORDINE.
0264 035F 60                                RTS      : RITORNO.
0265 0360 8D 00 40 HIBYTE STA PORT1B         : MEMORIZZA A NEI LED DI ORDINE
                                           : ELEVATO.
0266 0363 60                                RTS      : RITORNO.
0267 0364                                     :
0268 0364                                     : SUBROUTINE PER LA GENERAZIONE DEL SUONO.
0269 0364                                     :
0270 0364 85 04 TONE STA FREQ
0271 0366 A9 FF LDA # $ FF
0272 0368 8D 00 AC STA PORT3B
0273 036B A9 00 LDA # 00

0274 036D A6 03 LDX DUR
0275 036F A4 04 FL2 LDY FREQ
0276 0371 88 FL1 DEY
0277 0372 18 CLC
0278 0373 90 00 BCC *+ 2
0279 0375 D0 FA BNE FL1
0280 0377 49 FF EOR # $ FF
0281 0379 8D 00 AC STA PORT3B
0282 037C CA DEX
0283 037D D0 F0 BNE FL2
0284 037F 60 RTS

0285 0380                                     :
0286 0380                                     : SUBROUTINE DEL GENERATORE DI NUMERI CASUALI.
0287 0380                                     :
0288 0380 38 RANDOM SEC
0289 0381 A5 15 LDA RND+1
0290 0383 65 18 ADC RND+4
0291 0385 65 19 ADC RND+5
0292 0387 85 14 STA RND
0293 0389 A2 04 LDX # 4
0294 038B B5 14 RND SH LDA RND,X
0295 038D 95 15 STA RND+1,X
0296 038F CA DEX
0297 0390 10 F9 BPL RND SH
0298 0392 60 RTS
0299 0393 .END

SYMBOL TABLE
SYMBOL VALUE
CNTLP 02B3 DDR1A A003 DDR1B A002 DDR3B AC02
DELAY 0330 DISPLY 0227 DL1 0332 DL2 0334
DUR 0003 EVAL 02A7 FALL 0326 FL1 0371
FL2 036F FREQ 0004 GETKEY 0100 GETRND 0231
HIBYTE 0360 HILIM 0087 HITONE 0020 INCR 000B
INDX 0008 KEY 0218 LDRND 022F LEDUPD 0270
LIGHT 033D LOBYTE 035C LOLIM 005A LOSE 030A
LOSEND 031B LOTONE 00F0 LTABLE 001A LTCC 0355
LTMSK 000E LTSHFT 034D NORST 0258 NXTUPD 028F
OFFLD9 0280 PORT1A A001 PORT1B A000 PORT3B AC00
RANDOM 0380 RISE 0300 RND 0014 RND SH 038B
SCORE 0002 SCORLP 02BC SCORTB 0023 SCORTP 0001
SPDPRM 0050 SPDUPD 0269 SPEEDS 0005 START 0210
T1CL A004 TEMP 0000 TONE 0364 UPDATE 0245
UPDTLP 0247 VALUES 0011 WAIT 0294 WIN 02D6
WINEND 02EE
END OF ASSEMBLY

```

Figura 7.9: Programma della slot machine (continua)

## Variabili in pagina zero

In memoria sono immagazzinate le seguenti variabili:

- TEMP è una locazione dello scratch
- SCORTP viene impiegata per la memorizzazione temporanea del punteggio guadagnato o perso ad ogni rotazione
- SCORE è il punteggio del gioco
- DUR e FREQ specificano le solite costanti per la generazione del suono
- SPEEDS (3 locazioni) specifica la velocità di rotazione delle tre colonne
- INDX (3 locazioni): contatori di ritardo per le rotazioni dei LED
- INCR (3 locazioni): puntatori alle posizioni dei LED di ogni colonna, impiegati per l'esecuzione delle configurazioni d'uscita delle tabelle
- LTMSK (3 locazioni): configurazioni indicanti i LED accesi
- VALUES (3 locazioni): numero di LED accesi in ogni colonna
- RND (6 locazioni): scratch-pad per il generatore di numeri casuali.

## Implementazione del programma

Il programma consiste di un programma principale e di due subroutine principali: DISPLAY ed EVAL. Inoltre esso contiene alcune subroutine di utilità: DELAY per la generazione di un ritardo di lunghezza variabile, LIGHT per accendere il LED appropriato, TONE per generare un suono di un certo tono e RANDOM per generare un numero casuale.

Il programma principale è memorizzato alle locazioni di memoria 200 e successive. Come al solito, i tre registri di direzione dati per le Porte A e B del VIA #1 e della Porta B del VIA #3, devono essere condizionati come uscite:

```
LDA #$FF
STA DDRIA
STA DDRIB
STA DDR3B
```

Come nei capitoli precedenti, il registro contatore del timer 1 viene impiegato per fornire un numero casuale iniziale (un seme per il generatore di numeri casuali). Questo seme viene memorizzato alla locazione di memoria RND + 1, dove sarà utilizzato successivamente dalla subroutine di generazione di numeri casuali:

```
LDA TICL
STA RND + 1
```

All'inizio di un nuovo gioco, il punteggio iniziale è posto ad 8. Questo viene imposto mediante:

```
START      LDA #8  
           STA SCORE
```

e visualizzato:

```
           TAY          Y deve contenerlo  
           JSR LIGHT
```

La subroutine LIGHT viene impiegata per visualizzare il punteggio accendendo il LED corrispondente ai contenuti del registro Y. Questo sarà descritto in seguito.

Il programma della slot machine è ora pronto per rispondere al giocatore. Qualsiasi tasto può essere premuto:

```
KEY        JSR GETKEY
```

Non appena un tasto è stato premuto, le ruote devono essere fatte girare:

```
           JSR DISPY
```

Una volta che le ruote si sono fermate, deve essere valutato e visualizzato il punteggio con il relativo accompagnamento sonoro:

```
           JSR EVAL
```

Se il punteggio finale non è "0", il processo viene ri-inizializzato:

```
           LDA SCORE  
           BNE KEY
```

e l'utente può ancora far ruotare le ruote. Altrimenti, se il punteggio era "0", si inizia un nuovo gioco:

```
           BEQ START
```

Questo completa il corpo del programma principale. È abbastanza semplice perché già strutturato sotto forma di subroutine.

## Le subroutine

Gli algoritmi corrispondenti alle due subroutine principali DISPLAY ed EVAL sono stati descritti al paragrafo precedente. Consideriamo ora la relativa implementazione del programma.

### Subroutine DISPLAY

I tre parametri fondamentali di questa subroutine sono: LOLIM, HILIM ed SPDRPM. Per esempio, l'abbassamento di LOLIM condurrà ad un maggiore tempo di rotazione dei LED. Variando questi tre parametri è possibile ottenere diversi altri effetti. Si potrebbe ottenere una vincita quasi ogni volta! Ecco  $LOLIM = 90$ ,  $HILIM = 134$ ,  $SPDRPM = 80$ .

La locazione di memoria INCR viene impiegata come puntatore alla posizione di LED corrente. Essa verrà impiegata successivamente per prelevare l'appropriata configurazione di bit dalla tabella e può assumere i valori 0, 1 oppure 2 (puntando alle posizioni di LED 1, 2 oppure 3). I tre puntatori di ogni colonna dei LED sono immagazzinati rispettivamente alle locazioni di memoria INCR,  $INCR + 1$  ed  $INCR + 2$ . Essi devono essere inizializzati a 0:

```
DISPLY      LDA # 0
            STA INCR
            STA INCR + 1
            STA INCR + 2
```

Notate che, negli esempi precedenti (come in Fig. 7.7), per semplificare l'esposizione, abbiamo impiegato i puntatori X ed Y per rappresentare i valori tra 1 e 3. Adesso, invece, X e Y assumono valori compresi tra 0 e 2 in modo da facilitare l'indicizzazione. Il puntatore della ruota viene settato alla ruota di estrema destra:

```
LDRND      LDY #2
```

Dalla subroutine RANDOM si ottiene un numero casuale iniziale:

```
GETRND     JSR RANDOM
```

Il numero restituito dalla subroutine viene confrontato con i limiti inferiore e superiore di accettabilità. Se esso non appartiene all'intervallo specifico, viene respinto e si ottiene un nuovo numero finchè non se ne trova uno che appartiene all'intervallo richiesto.



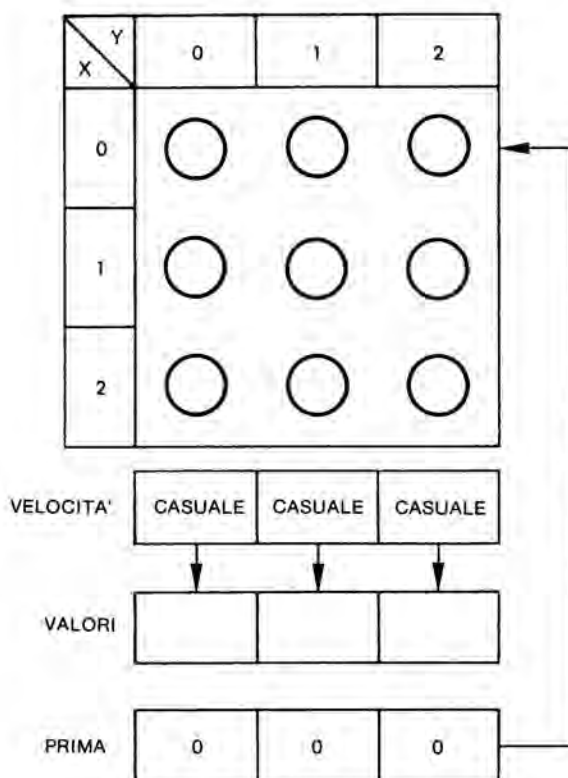


Figura 7.10: Rotazione delle ruote

CMP #HILIM	Troppo grande?
BCS GETRND	Se si, accettane un altro
CMP # LOLIM	Troppo piccolo?
BCC GETRND	Se si, accettane un altro

Il numero casuale valido viene quindi memorizzato nella locazione indice INDX e nella locazione SPEEDS della colonna corrente. (Vedere la Fig. 7.10).

```
STA INDX,Y
STA SPEEDS,Y
```

Lo stesso processo viene eseguito per le colonne 1 e 0:

DEY		
BPL GETRND		Accetta il numero casuale successivo

Una volta ottenuti l'indice e la velocità di tutte e tre le colonne, si inizia un nuovo ciclo di iterazione, impiegando il registro X come contatore della ruota:

UPDATE	LDX #2	Poni il contatore per tre iterazioni
--------	--------	--------------------------------------

Si esegue il test della velocità rispetto al valore 0:

UPDTLP	LDY SPEEDS,X	Velocità (X) = 0?
	BEQ NXTUPD	Se così aggiorna la colonna successiva

Finchè la velocità non è zero, occorre accendere il LED successivo di quella colonna. Il conteggio del ritardo viene decrementato;

DEC INDX,X	Ciclo di decremento, indice (X)
------------	---------------------------------

Se il ritardo non è stato decrementato a zero, si ha una ramificazione a NXTUPD che verrà descritta più avanti. Altrimenti, se il contatore di ritardo INDX è decrementato a 0, occorre accendere il LED successivo. Il puntatore del LED viene incrementato oppure riparte dall'inizio se esso raggiunge il valore 3:

	BNE NXTUPD	Se indice del ciclo (X) < > 0, esegui l'aggiornamento successivo
	LDY INCR,X	Incrementa puntatore
	INY	
	CPY #3	Puntatore = 3?
	BNE NORST	Se no, salta
	LDY #0	Reset a 0
NORST	STY INCR,X	Rimemorizza puntatore (X)

Il nuovo valore del puntatore del LED viene ri-memorizzato in INCR per la colonna appropriata. (Ricordate che all'interno della routine UPDATE, X punta alla colonna). Per accendere il LED appropriato, occorre ricavare una configurazione di bit da LTABLE. Notate che LTABLE (ed anche SCORTB) viene trattata dal punto di vista concettuale, come se si trattasse di una matrice bi-dimensionale, cioè avente

righe e colonne. Tuttavia, sia LTABLE che SCORTB, sono immagazzinate in memoria come una serie continua di numeri. Quindi, per ottenere l'indirizzo di un particolare elemento, il numero di riga deve essere moltiplicato per il numero di colonne e quindi sommato al numero di colonne.

Si accederà alla tabella con l'impiego del modo di indirizzamento indicizzato, in cui il registro Y viene impiegato come indice. Per accedere alla tabella, X deve essere preliminarmente moltiplicato per 3 e poi occorre sommarli il valore di INCR (cioè il puntatore del LED).

La moltiplicazione per 3 viene eseguita per mezzo di uno scorrimento a sinistra seguito da un'addizione, essendo lo scorrimento a sinistra equivalente ad una moltiplicazione per 2:

STX TEMP	Moltiplica X per 3
TXA	
ASL A	Scorrimento a sinistra
CLC	
ADC TEMP	Più uno

Viene quindi sommato il valore di INCR ed il totale è trasferito nel registro Y in modo da impiegare la tecnica di indirizzamento indicizzato. Infine si può prelevare l'ingresso da LTABLE:

ADC INCR,X	
TAY	
LDA LTABLE,Y	Accetta la struttura di LED

Una volta che la configurazione è stata ottenuta, viene memorizzata in una delle tre locazioni di memoria all'indirizzo LTMSK e successivi. La configurazione viene memorizzata alla locazione di memoria corrispondente alla colonna che viene correntemente aggiornata, dove il LED è stato "mosso". L'accensione dei LED deve avvenire soltanto dopo che è stato implementato il pattern completo di tutte e tre le colonne. In conseguenza dell'avanzamento della posizione del LED all'interno della colonna, la costante di velocità deve essere incrementata.

	STA LTMSK,X
SPDUPD	LDY SPEEDS,X
	INY
	STY SPEEDS,X

L'indice viene posto uguale alla nuova velocità:

STY INDX,X

Notate che ora sarà necessaria una manipolazione speciale per il LED #9. Il pattern da visualizzare sui primi otto LED era memorizzato in LTABLE. Si riconosce facilmente che il LED # 9 deve essere acceso dal fatto che la configurazione della colonna #3 mostra tutti zeri: poichè un LED deve essere sempre acceso in una colonna, questo implica che occorre accendere il LED #9:

```
LEDUPD    LDA  #0
           STA  PORT1B      Reset del LED 9
```

Successivamente si ottiene il pattern della terza colonna dalla locazione in cui è stata salvata in LTMSK + 2. Si esegue il test rispetto al valore 0:

```
LDA LTMSK + 2
BNE OFFLD9
```

Se questa configurazione è zero, allora occorre accendere il LED #9:

```
LDA #01
STA PORT1B
```

Altrimenti si ha una ramificazione alla locazione OFFLD9 e verranno accesi i LED rimanenti. La struttura ora contenuta nell'accumulatore ed ottenuta da LTMSK + 2 viene successivamente messa in OR con le configurazioni della seconda e terza colonna:

```
OFFLD9    LDA #0
           ORA LTMSK
           ORA LTMSK + 1
```

A questo punto, A contiene la configurazione finale che deve essere inviata alla porta d'uscita per attivare la configurazione di LED richiesta. Si ha quindi:

```
STA PORT1A
```

E, contemporaneamente, viene attivato l'altoparlante:

```
LDA PORT3B
EOR #$FF
STA PORT3B
```

È importante capire che, anche se sono stati mossi solo i LED di una delle tre colonne, è necessario accendere contemporaneamente i LED di

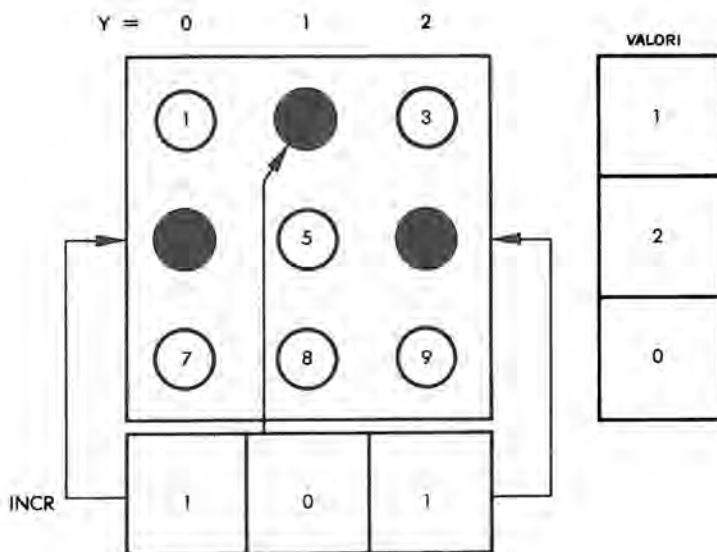


Figura 7.11: Valutazione della fine di una rotazione

tutte le colonne oppure la prima e la seconda colonna si spegneranno.

Una volta considerata la terza colonna, occorre passare alla successiva. Il puntatore di colonna X viene perciò decrementato ed il processo continua:

```

NDTUPD    DEX
           BPL UPDTLP      Se X >= 0 esegui
                           l'aggiornamento successivo

```

Una volta che sono state manipolate la prima e la seconda colonna, viene implementato un ritardo per evitare un lampeggio troppo veloce dei LED. Questo ritardo viene controllato dal parametro di velocità SPDPRM:

```

           LDY #SPDPRM
WAIT       DEY
           BNE WAIT

```

Una volta eseguito questo ciclo completo, si controlla se la locazione della velocità di ogni colonna contiene 0. Se tutte le colonne sono zero, la rotazione è conclusa:

```

- LDA SPEEDS
  ORA SPEEDS + 1
  ORA SPEEDS + 2
  BNE UPDATE

```

Altrimenti, si ha una ramificazione alla locazione UPDATE. Se tutti i LED si sono arrestati, deve essere generata una pausa in modo che l'utente possa vedere la struttura:

```

LDA #$FF
STA DUR
JSR DELAY

```

e quindi si esce:

```

RTS

```

*Esercizio 7-2: Notate che i contenuti delle tre locazioni SPEEDS sono stati posti in OR per eseguire il test se sono tre zeri. Sarebbe stato equivalente sommarli assieme?*

### Subroutine EVAL

Questa subroutine costituisce l'interfaccia d'uscita dell'utente. Essa calcola il punteggio ottenuto dal giocatore e genera gli effetti visivi e sonori. Le costanti per le frequenze del tono acuto generato da una situazione perdente sono specificate all'inizio di questa subroutine:

```

HITONE = $20
LOTONE = $F0

```

Il metodo impiegato per calcolare il numero di LED accesi per ogni riga è stato discusso e mostrato in Fig. 7.7. Il numero di LED accesi per ogni riga è inizialmente posto a zero:

```

EVAL      LDA #0
          STA VALUES
          STA VALUES + 1
          STA VALUES + 2

```

Anche il punteggio temporaneo viene posto a zero:

```

STA SCORTP

```

Il registro indice Y verrà impiegato come puntatore di colonna, e verrà calcolato il numero di LED accesi per ogni riga. Il numero di LED accesi per la colonna corrente viene ottenuto leggendo l'ingresso appropriato INCR. Osservata l'esempio in Fig. 7.11. Il valore contenuto in ciascuna delle tre locazioni INCR è il numero di riga. Questo numero di riga viene memorizzato nel registro X ed impiegato come indice per incrementare il valore appropriato nella tabella VALUES. Notate che tutto viene eseguito in appena due istruzioni, chiaramente impiegando due volte la tecnica di indirizzamento indicizzato del 6502:

```
CNTLP      LDY #2           3 iterazioni
            LDX INCR,Y
            INC VALUES,X
```

Questo viene eseguito una volta per la colonna 2, il processo viene quindi ripetuto per le colonne 1 e 0:

```
DEY
BPL CNTLP
```

A questo punto verrà eseguita un'altra iterazione per convertire i numeri finali caricati nella tabella VALUES in punteggi reali sulla base delle specifiche della tabella dei punteggi, SCORTB. Il registro indice X viene impiegato come puntatore di riga per VALUES e SCORTB.

```
LDX #2
```

Poichè la tabella SCORTB ha quattro ingressi di un byte per il livello di riga, per accedere al byte corretto all'interno della tabella il numero di riga deve essere preliminarmente moltiplicato per 4, quindi occorre sommargli il "valore" corrispondente (numero di LED accesi). La moltiplicazione per 4 viene implementata con due scorrimenti successivi a sinistra:

```
SCORLP     TXA
            ASL A
            ASL A
```

Il numero attualmente contenuto nell'accumulatore è uguale a 4 volte il valore contenuto in X, cioè 4 volte il valore del puntatore di riga. Per ottenere l'offset (fuori zero) finale per la tabella SCORTB, dobbiamo sommare a questo valore il numero di LED accesi in quella riga, cioè il numero contenuto nella tabella VALUES. Come al solito, questo numero viene ottenuto eseguendo un'operazione di indirizzamento indicizzato:

CLC  
ADC VALUES, X      Indirizzo di colonna nell'array

Si ottiene in tal modo l'offset finale corretto per l'accesso a SCORTB.

Ora si può eseguire l'accesso indicizzato alla tabella SCORTB. Il registro indice Y viene impiegato per questo scopo ed i contenuti nell'accumulatore vengono trasferiti in esso:

TAY

Viene eseguito l'accesso:

LDA SCORTB, Y      Accetta il punteggio di questa  
rotazione

Il punteggio corretto corrispondente al numero di LED accesi nella riga puntata dal registro indice X è ora contenuto nell'accumulatore. Il punteggio parziale ottenuto per la riga corrente viene ora sommato al totale corrente per tutte le righe:

CLC  
ADC SCORTP      Totale dei punteggi  
STA SCORTP      Salvato

Viene quindi decrementato il numero di riga in modo da poter esaminare la riga successiva. Se viene eseguito il decremento di X dal valore 0, cioè se si ottiene un valore negativo, si esce; altrimenti si riesegue il ciclo:

DEX  
BPL SCORLP

A questo punto è stato ottenuto il punteggio totale della rotazione corrente. Occorre segnalare al giocatore, in modo visivo ed udibile una vittoria oppure una sconfitta. Per preparare l'attivazione dell'altoparlante, la locazione di memoria DUR viene impostata alla durata corretta del tono:

LDA #\$60  
STA DUR

Viene quindi esaminato il punteggio: se è 0 si ha una ramificazione alla routine LOSE:

LDA SCORTP  
BEQ LOSE

Altrimenti si tratta di una vittoria. Esaminiamo queste due routine.



### *Routine WIN*

Il punteggio finale (per tutte le rotazioni del gioco) è contenuto nella locazione di memoria SCORE (punteggio). Questa locazione di memoria verrà incrementata di un punto alla volta e controllata rispetto al massimo valore 16 ogni volta. Eseguiamo:

```
WIN          INC SCORE
             LDY SCORE
             CPY #16
```

Se è stato raggiunto il massimo valore 16, è la fine del gioco e si ha una ramificazione alla locazione WINEND:

```
BEQ WINEND
```

Altrimenti viene aggiornata la visualizzazione del punteggio ed emesso un beep:

```
JSR LIGHT
```

La routine LIGHT verrà descritta in seguito. Essa visualizza il punteggio del giocatore. Successivamente viene emesso un beep:

```
LDA #HITONE
JSR TONE
```

Anche la routine TONE verrà descritta in seguito.

Viene quindi implementato un ritardo:

```
JSR DELAY
```

quindi viene decrementato il punteggio di questa rotazione:

```
DEC SCORTP
```

e controllato rispetto al valore zero. Se è 0, il gioco è terminato, altrimenti viene rieseguito il ciclo:

```
BNE WIN
RTS
```

### *Routine WINEND*

Si accede a questa routine ogni volta che si raggiunge un punteggio totale pari a 16. Questa è la fine del gioco. Tutti i LED vengono accesi

contemporaneamente e viene attivato un suono di sirena con frequenza crescente. Infine si ha un restart del gioco.

Tutti i LED vengono accesi caricando la struttura appropriata nella Porta 1A e Porta 1B:

LDA #\$FF	
STA PORT1A	Accendi tutti i LED
STA PORT1B	

Le variabili vengono ri-inizializzate: il punteggio totale diventa zero, il che segnala al programma principale l'inizio di un nuovo gioco, la locazione di memoria DUR viene posta a 4 per controllare la durata del tempo di esecuzione dei beep ed il parametro di frequenza viene posto ad "FF" alla locazione TEMP:

STA TEMP	Parametro di frequenza
LDA #0	
STA SCORE	Azzera per il restart
LDA #4	
STA DUR	Durata del beep

Per generare un beep viene impiegata la subroutine TONE:

RISE	LDA TEMP	Accetta la frequenza
	JSR TONE	Genera il beep

La costante di frequenza del beep viene quindi decrementata e viene eseguito il beep successivo ad un'altezza leggermente più alta:

```
DEC TEMP
BNE RISE
```

Ogni volta che la costante di frequenza viene decrementata a 0, la sirena è completa e si esce dalla routine:

```
RTS
```

### *Routine LOSE*

Ora esaminiamo cosa succede nel caso di una situazione di sconfitta. Gli eventi sono essenzialmente simmetrici rispetto a quelli descritti nel caso di una vittoria.

Nel caso di una sconfitta, il punteggio deve essere aggiornato solo una

volta. Esso viene decrementato di uno:

```
LOSE      DEC SCORE
```

Il punteggio diminuito viene visualizzato al giocatore:

```
LDY SCORE  
JSR LIGHT
```

Viene generato un suono udibile:

```
LDA #LOTONE  
JSR TONE
```

Viene controllato il valore finale del punteggio per vedere se è stato raggiunto un punteggio "0". In caso affermativo il gioco è terminato; altrimenti viene iniziata la rotazione successiva;

```
LDY SCORE  
BEQ LOSEND  
RTS
```

Osserviamo che cosa succede quando si raggiunge un punteggio "0" (LOSEND). Verrà generata una sirena con frequenza decrescente. Tutti i LED saranno spenti sulla scheda;

```
LOSEND    LDA #0  
          STA TEMP  
          STA PORT1A      Spegni il LED #1
```

La durata del beep per ogni frequenza viene posta al valore 4, memorizzato nella locazione di memoria DUR:

```
LDA #4  
STA DUR
```

Viene quindi generato il beep alla frequenza corrente:

```
FALL      LDA TEMP  
          JSR TONE      Esegui il beep
```

Successivamente la costante di frequenza viene aumentata di 1 ed il processo viene ri-inizializzato fino all'overflow del registro TMP.

```
INC TEMP      Il tono successivo sarà  
              più basso
```

## BNE FALL RTS

Questo completa la nostra descrizione del programma principale. Esaminiamo ora le quattro subroutine che vengono impiegate. Queste sono: DELAY, LIGHT, TONE e RANDOM.

### *Subroutine DELAY*

Questa subroutine implementa un ritardo; la durata del ritardo viene imposta dal contenuto della locazione di memoria DUR. La lunghezza del ritardo risultante sarà uguale a  $(2046 \times \text{DUR} + 10)$  microsecondi. Il ritardo viene implementato impiegando una scrittura convenzionale a cicli annidati a due livelli. Il ciclo di ritardo più interno viene controllato dal registro indice X, mentre il ciclo di ritardo più esterno viene controllato dal registro indice Y, che viene inizializzato dal contenuto della locazione di memoria DUR. Quindi Y viene inizializzato:

```
DELAY      LDY DUR
```

Viene quindi implementato il ritardo del ciclo più interno:

DL1	LDX #\$FF	
DL2	BNE*+2	Consuma tempo
	DEX	Contatore del ciclo più interno
	BNE DL2	Ciclo interno

Infine viene implementato il ciclo esterno:

```
DEY  
BNE DL1  
RTS
```

**Esercizio 7-3:** Verificate l'esatta durata del ritardo implementato dalla subroutine DELAY.

### *Subroutine LIGHT*

Questa subroutine accende i LED corrispondenti al numero contenuto nel registro Y. Ricordate che i quindici LED sulla Scheda Giochi sono numerati esternamente da 1 a 15 ma sono connessi ai bit da 0 a 7 della Porta 1A e da 0 a 7 della Porta 1B. Quindi, se deve essere visualizzato un punteggio pari ad 1, deve essere posto ad 1 il bit 0 della Porta 1A. In

generale il bit N della Porta 1A deve essere posto ad 1 quando N è uguale al punteggio meno 1. Comunque esiste un'eccezione. Per vedere questo, fate riferimento alla Fig. 1.4 che mostra le connessioni dei LED. Notate che il bit 6 della Porta 1B non è connesso a nessun LED. Ogni volta che deve essere visualizzato un punteggio pari a quindici, vien posto ad 1 il bit 7 della Porta 1B. Questa eccezione verrà considerata nella routine semplicemente non decrementando il punteggio quando esso ammonta a 15.

La configurazione corretta di accensione dei LED appropriati si otterrà facendo scorrere un "1" nell'accumulatore in posizione corretta. In un esercizio successivo verranno suggeriti altri metodi. Preliminarmente eseguiamo l'inizializzazione:

```
LIGHT      LDA #0
            STA TEMP
            STA PORT1A
            STA PORT1B
```

Dobbiamo prima considerare la situazione in cui il punteggio contenuto in Y sia 15 ed, in questo caso, occorre non eseguire nulla (nessuno scorrimento):

```
CPY#15      Codice per bit non corretto?
BEQ*+3      Se sì, nessuna variazione
```

Per qualsiasi altro punteggio, viene eseguito prima il decremento e poi lo scorrimento:

```
            DEY      Decrementa al codice interno
            SEC      Poni ad 1 il bit di scorrimento
LTSHFT      ROL A
```

La prima istruzione di questa subroutine azzeri i contenuti dell'accumulatore. Il bit di riporto (carry) è settato al valore 1 e quindi fatto scorrere nella posizione di estrema destra di A. (Vedere Fig. 7.12). Questo processo viene ripetuto tante volte quante sono necessarie. Poichè dobbiamo contare da 1 a 14, oppure da 0 a 13, si verifica un overflow quando l'"1" fatto ruotare nell'accumulatore "cade fuori" a sinistra. Finchè questo non accade il processo di scorrimento continua e viene implementata una ramificazione alla locazione LTCC:

```
BCC LTCC
```

Comunque, se il bit "1" cade fuori a sinistra dell'accumulatore, il valore "FF" viene caricato alla locazione di memoria TEMP per segnalare

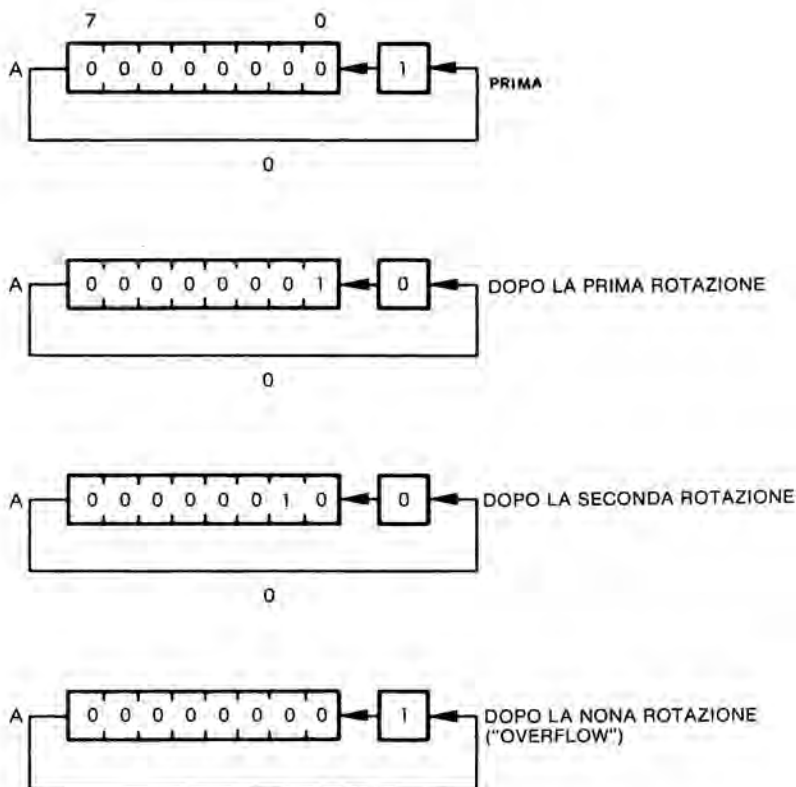


Figura 7.12: Generazione della struttura di LED

questa eventualità. Ricordate che il valore era azzerato dalla seconda istruzione della subroutine LIGHT.

```
LDX #$FF
STX TEMP
```

Il bit "1" viene quindi mosso dal carry alla posizione di estrema destra dell'accumulatore. Successivamente verrà controllato il valore contenuto nella locazione di memoria TEMP e questo determinerà se la struttura contenuta nell'accumulatore deve essere inviata alla Porta 1A oppure alla Porta 1B.

Il processo di scorrimento continua. Il contatore viene decrementato e, se esso raggiunge il valore "0", si esce; altrimenti il processo viene

ripetuto:

	ROL A
LTCC	DEY
	BPL LTSHFT

Una volta completato il processo, viene esaminato il valore della locazione di memoria TEMP. Se questo valore è "0", significa che non si è verificato alcun overflow e che deve essere impiegata la Porta 1A. Se questo valore non è "0", cioè se è "FF", allora occorre impiegare la Porta 1B:

	LDX TEMP	Accetta il flag di overflow
	BNE HIBYTE	
LOBYTE	STA PORT1A	Invio ai LED inferiori
	RTS	Ritorno
HIBYTE	STA PORT1B	Invio ai LED superiori
	RTS	

### *Subroutine TONE*

Questa subroutine genera un beep. La frequenza del beep è determinata dai contenuti dell'accumulatore al momento della chiamata; la durata del beep è impostata dai contenuti della locazione di memoria DUR. Questo è già stato descritto al Capitolo 2.

### *Subroutine RANDOM*

Si tratta di un semplice generatore di numeri casuali. La subroutine è già stata descritta al Capitolo 3.

**Esercizio 7-4:** *Suggerite un altro modo per generare la configurazione di LED corretta da caricare nell'accumulatore, senza l'impiego di una sequenza di rotazioni.*

### **Variazioni del gioco**

Le tre righe di LED disponibili sulla Scheda Giochi possono essere interpretate in un modo diverso da quello impiegato all'inizio di questo capitolo. La riga 1 potrebbe essere interpretata, diciamo, come ciliegie, la riga 2 come stelle e la riga 3 come arance. Quindi un LED acceso nella riga 1 alla fine della rotazione rappresenta una ciliegia, mentre due LED

nella riga 3 mostrano due arance. La combinazione risultante è una ciliegia e due arance. La tabella del punteggio impiegata in questo programma può essere alternata per assegnare un punteggio differente per ogni combinazione, in dipendenza del numero di ciliegie, arance o stelle presenti alla fine della rotazione. A questo punto è molto semplice modificare i valori caricati nella tabella del punteggio. Quando vengono caricati nuovi valori nella tabella del punteggio, verrà implementato un risultato di assegnazione del punteggio completamente diverso. Non saranno necessarie altre variazioni del programma.

## SOMMARIO

Questo programma, benchè apparentemente semplice, è relativamente complesso, e può portare a giochi notevolmente diversi a seconda della formula di valutazione impiegata al momento dell'arresto della rotazione. Per motivi di chiarezza, il programma è stato organizzato in routine separate che possono essere studiate individualmente.



## CAPITOLO 8

# ECO

### LE REGOLE

Lo scopo di questo gioco è riconoscere e duplicare una sequenza di luci e suoni che vengono generati dal computer. Alcune variazioni di questo gioco, come "Simon" e "Follow me" (marchi registrati\*) sono commercialmente disponibili. In questa versione il giocatore deve specificare, prima di iniziare il gioco, la lunghezza della sequenza da riconoscere. Il giocatore indica la sua preferenza di lunghezza premendo il tasto opportuno tra 1 e 9. A questo punto il computer genera una sequenza casuale avente la lunghezza desiderata. Questa sequenza può essere ascoltata e visualizzata premendo uno qualsiasi dei tasti alfabetici (dalla A alla F).

Quando viene premuto uno dei tasti alfabetici, la sequenza generata dal programma viene visualizzata sui LED corrispondenti (contrassegnati da 1 a 9) sulla Scheda Giochi, con contemporanea esecuzione all'altoparlante di una sequenza di note. Il giocatore dovrebbe prestare molta attenzione ai suoni e/o alle luci e, successivamente introdurre la sequenza di numeri che ha identificato. Ogni volta che il giocatore preme il tasto corretto, il LED corrispondente sulla Scheda Giochi si accende, indicando un successo. Ogni volta che viene commesso un errore, viene emesso un tono basso.

Alla fine del gioco, se il giocatore ha indovinato si accendono tutti i LED sulla Scheda Giochi e viene eseguita una scala crescente (successione di note). Se il giocatore non ha indovinato, sulla Scheda Giochi si accenderà soltanto un LED indicante il numero di errori commessi e verrà seguita una scala decrescente.

---

\* "Follow me" è un marchio registrato della Atari, Inc., "Simon" è un marchio registrato della Mil-Bradley Co.

Se il giocatore ha indovinato, il gioco viene ri-inizializzato, altrimenti viene azzerato il numero di errori ed il giocatore ha un'altra possibilità di indovinare la serie.

In qualsiasi istante durante il gioco, il giocatore può premere uno dei tasti alfanumerici che gli consente di ascoltare ancora la sequenza. Tutti i tentativi precedenti vengono cancellati ed il gioco ricomincia.

Due LED nella riga in basso della matrice di LED vengono impiegati per comunicare con il giocatore.

Il LED 10 (il LED di estrema sinistra) indica "computer pronto - carica la lunghezza desiderata della sequenza".

Il LED 11 si accende immediatamente dopo che il giocatore ha specificato la lunghezza della sequenza. Esso rimane acceso durante il gioco e significa "introduci il tuo tentativo".

A questo punto il giocatore ha 3 possibilità:

1. Premere un tasto corrispondente al numero della sequenza che tenta di riconoscere.

2. Premere il tasto 0. Questo impone la ri-inizializzazione del gioco.

3. Premere un tasto dalla A alla F. Questo impone al computer la ri-esecuzione della stessa sequenza e ri-inizializza la sequenza dei tentativi.

## **Variazioni**

Il programma fornisce un test molto valido per valutare le vostre capacità musicali. Vi suggeriamo di iniziare un nuovo gioco ascoltando la sequenza, senza osservare i LED. Questo perchè i LED sulla Scheda Giochi sono numerati ed è abbastanza facile ricordare la sequenza di accensione semplicemente memorizzando i numeri. Questo potrebbe essere troppo semplice. Inoltre dovrete iniziare con sequenze di una sola nota. Se avete successo continuate con una sequenza di due note e quindi con una sequenza di tre note. Sfidate altri giocatori, vince il giocatore che è in grado di riconoscere la sequenza più lunga. Notate che alcuni giocatori sono in grado di riconoscere abbastanza facilmente una sequenza di nove note.

Dopo l'esecuzione di un certo numero di note (per esempio quando sono state eseguite più di cinque note), in modo da facilitare il tentativo, si può consentire al giocatore di guardare i LED sulla Scheda Giochi. Un altro approccio potrebbe essere quello di consentire al giocatore di premere uno dei tasti alfabetici in qualsiasi momento in modo da ascoltare ancora la sequenza; potrebbe essere motivo di penalità. Questa potrebbe consistere nel richiedere al giocatore il riconoscimento di una seconda sequenza della stessa lunghezza prima di passare ad una più lunga. Ciò significa che, per esempio, se un giocatore tenta di riconoscere

una sequenza di nove note ma si innervosisce dopo aver commesso un errore e dimentica la sequenza, si può consentire al giocatore di premere uno dei tasti alfabetici e riascoltare la sequenza. Comunque, se il giocatore riesce al secondo tentativo, egli deve successivamente riconoscere un'altra sequenza di nove note prima di procedere ad una di dieci note.

Questo metodo può essere esteso consentendo a qualsiasi giocatore di ripetere la configurazione memorizzata fino ad un massimo di due, tre, o cinque volte per gioco. In altre parole, nel corso del gioco, un giocatore può ripetere la sequenza premendo uno dei tasti alfabetici, ma questa possibilità non può essere sfruttata più di  $n$  volte.

### *Un ESP tester*

Un'altra variazione di questo gioco è di tentare di riconoscere la sequenza senza ascoltarla o vederla! Chiaramente in questo caso potete basarvi sui vostri poteri ESP (Extra Sensory Perception: percezione extra sensoriale) in modo da facilitare il tentativo. Per determinare se avete dei poteri ESP oppure no, ponete la lunghezza iniziale della sequenza ad "1". Quindi premete il tasto per tentare di individuare la nota selezionata dal programma. Ripetete numerose volte. Se non avete poteri ESP i vostri risultati dovrebbero essere casuali. Statisticamente dovrete vincere una volta su nove, che è soltanto un nono del tempo, ovvero l'11,11% del tempo. Notate che questa percentuale è valida solo per un numero molto elevato di tentativi.

Se vincete più dell'11% del tempo, potete avere dei poteri ESP! Se il vostro punteggio è maggiore del 50% intraprendete subito una carriera politica oppure dedicatevi agli affari. Se il vostro risultato è minore dell'11% siete "ESP negativo" ed è meglio che guardiate attentamente da entrambe le parti prima di attraversare la strada.

Il seguente esercizio è dedicato ai lettori che hanno delle basi di statistica.

**Esercizio 8-1:** *Calcolate la probabilità statistica di indovinare un tentativo su una sequenza di due numeri e su una sequenza di quattro numeri.*

### **UNA GIOCATA TIPICA**

Il programma inizia alla locazione 200. Come appare in Fig. 8-1, si accende il LED 10. Specifichiamo una serie di lunghezza due premendo il tasto "2" sulla tastiera. La visualizzazione sui LED riportata in Fig. 8-2 significa "introduci il tuo tentativo".

Vogliamo ascoltare l'accordo e quindi premiamo il tasto "F". In risposta i LED 5 e 2 si accendono brevemente sulla Scheda Giochi e le

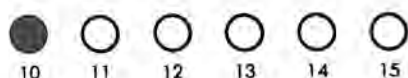


Figura 8.1: Specifica della lunghezza della sequenza da duplicare

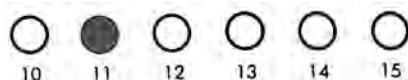


Figura 8.2: Caricate il vostro tentativo

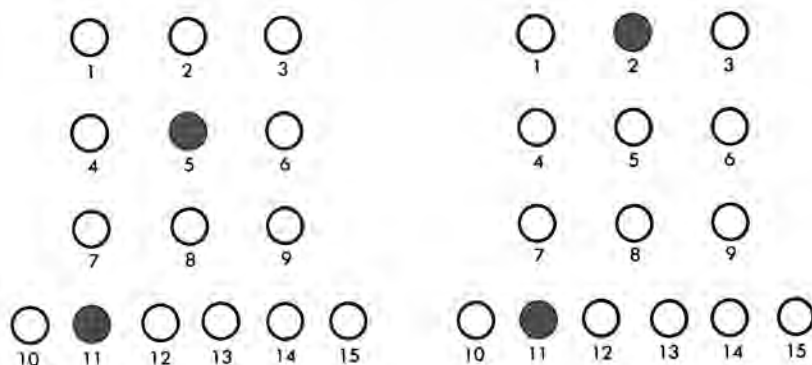


Figura 8.3: Seguitemi

note corrispondenti vengono eseguite attraverso l'altoparlante. Questo è illustrato in Fig. 8.3. Ora dobbiamo caricare la sequenza che abbiamo riconosciuto. Premiamo "5" sulla tastiera. In risposta il LED 11 si accende mentre il 5 si accende brevemente. Contemporaneamente la nota corrispondente viene eseguita attraverso l'altoparlante. È stato un tentativo riuscito!

Quindi premiamo il tasto "2". Il LED 2 si accende e l'altoparlante produce la nota corrispondente indicando che anche il secondo tentativo ha avuto successo. Un istante dopo tutti i LED sulla scheda si accendono in segno di congratulazioni e viene suonata una scala crescente. Si tratta di una sequenza di note di frequenza crescenti che confermano la vitto-

ria. Il gioco viene quindi ri-inizializzato ed il LED 10 si accende come mostrato in Fig. 8.1.

Seguiamo ora una sequenza perdente: il LED 10 è illuminato all'inizio del gioco, come in Fig. 8.1. Questa volta premiamo il tasto "1" per specificare una sequenza di una nota. Il LED 11 si accende, come mostrato in Fig. 8.2. Premiamo il tasto "F" e la nota viene eseguita sull'altoparlante. (Non guardiamo sulla Scheda Giochi il LED che si è acceso perchè sarebbe troppo semplice). Premiamo il tasto "3". Viene emesso un suono di sconfitta e si accende il LED 1 che indica l'esecuzione di un errore. Viene quindi eseguita una scala decrescente (note di frequenza decrescente) per confermare al giocatore sfortunato che il tentativo di indovinare la sequenza non è riuscito. Quindi il gioco continua con la stessa sequenza e lunghezza, cioè la situazione è ancora quella indicata in Fig. 8.2.

Se a questo punto il giocatore vuole cambiare la lunghezza della sequenza, oppure caricare una nuova sequenza, egli deve esplicitamente ri-inizializzare il gioco premendo il tasto 0. Dopo aver premuto il tasto 0, la situazione sarà quella indicata in Fig. 8.1, dove è possibile nuovamente specificare la lunghezza della sequenza.

## L'ALGORITMO

La Fig. 8.4 riporta il diagramma di flusso di questo programma. Esaminiamolo passo per passo:

1. Il programma dice al giocatore di selezionare una lunghezza della sequenza mediante accensione del LED 10 sulla Scheda Giochi.
2. La lunghezza della sequenza viene letta dalla tastiera. (A questo punto i tasti 0 ed A-F vengono ignorati.)
3. Le due variabili principali sono inizializzate a "0", cioè vengono azzerati il numero di tentativi ed il numero di errori.
4. Deve quindi essere generata una tabella di sequenza della lunghezza richiesta. La generazione avviene per mezzo di numeri casuali compresi tra 1 e 9.
5. Successivamente viene acceso il LED 11 e letto il tasto premuto dal giocatore.
6. Se è "0" il gioco viene ri-inizializzato. Altrimenti si continua.
7. Se il valore del tasto premuto è maggiore o uguale a 10, si tratta di un carattere alfabetico e si salta alla parte destra del diagramma di flusso ai passi 8 e 9. La sequenza registrata viene visualizzata al giocatore, tutte le variabili sono ri-inizializzate a zero e si ricomincia il processo di tentativi. Se il tasto premuto era un numero compreso tra 1 e 9, occorre confrontarlo con il valore memorizzato. Si va a 10 sul diagramma di flusso.

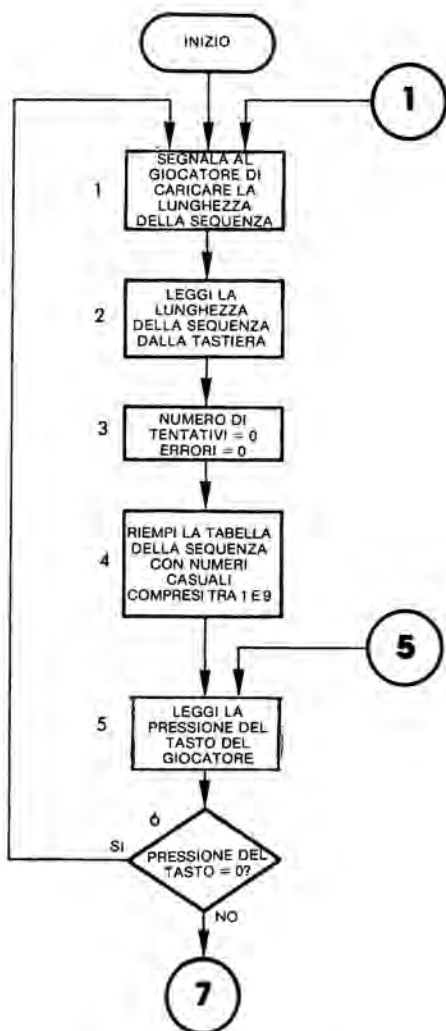


Figura 8.4: Diagramma di flusso dell'Eco

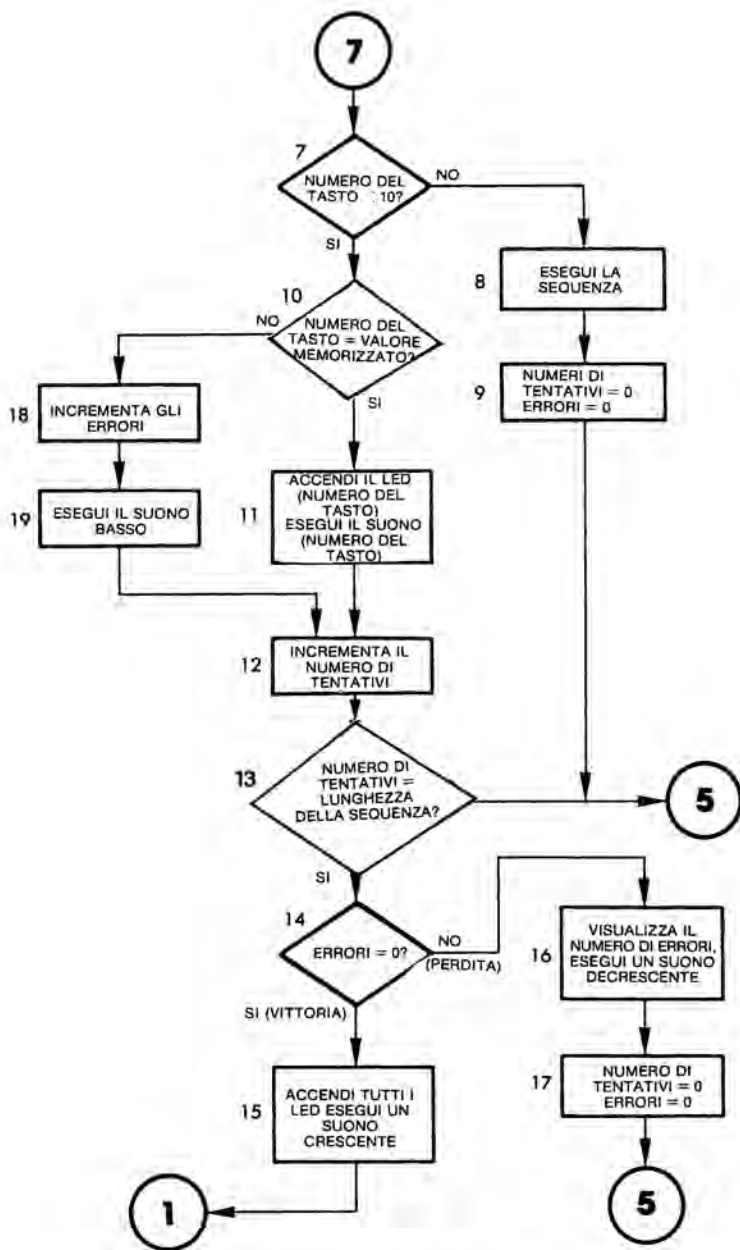


Figura 8.4: Diagramma di flusso dell'Eco (continua)

10. Se il tentativo era corretto, si salta a destra sul diagramma di flusso al passo 11.
11. Poichè il tasto premuto corrisponde al valore memorizzato in memoria, viene acceso il LED corrispondente sulla Scheda Giochi e viene eseguita la nota corrispondente al tasto premuto.
12. Il numero di tentativi viene incrementato e confrontato con la lunghezza massima della sequenza da indovinare.
13. Viene eseguito un controllo per vedere se è stata raggiunta la massima lunghezza della sequenza. In caso negativo si salta indietro al passo 5 sul diagramma di flusso e si legge il valore del tasto successivamente premuto. Se è stata raggiunta la massima lunghezza della sequenza, procediamo in basso nel diagramma di flusso al blocco contrassegnato con 14.
14. Viene controllato il numero totale di errori fatti dal giocatore. Viene eseguito il confronto della variabile ERRORS con il valore "0". Se essa è "0" si ha una situazione vincente e si salta al blocco 15.
15. Tutti i LED della scheda vengono accesi, viene eseguita una sequenza di note in ordine crescente e si salta indietro all'inizio del gioco.

Ritorniamo indietro al blocco 14. Se il numero di errori era maggiore di zero, si ha una situazione di "sconfitta" e si salta al blocco 16.

16. Viene visualizzato il numero di errori e vengono generate le note in ordine decrescente.
17. Tutte le variabili sono ri-inizializzate a zero e si salta al blocco 5, fornendo al giocatore un'altra possibilità per indovinare la serie.

Ora torneremo a dedicare la nostra attenzione al blocco 10 del diagramma di flusso, dove era stato controllato il valore del tasto rispetto al valore memorizzato. Assumeremo questa volta che il tentativo sia errato e si salta alla sinistra del blocco 10.

18. Il numero di errori eseguiti dal giocatore viene incrementato di uno.
19. Viene eseguita una nota bassa per indicare una situazione di sconfitta. Il programma quindi salta indietro al blocco 12 e procede come prima.

## IL PROGRAMMA

La Fig. 5.1 riporta il programma completo. Il programma impiega due tabelle e diverse variabili. Le due tabelle sono: NOTAB, impiegata per specificare le frequenze delle note e DURTAB, impiegata per specificare le durate delle note. Entrambe queste tabelle sono state introdotte al Capitolo 2. Esse non verranno descritte in questa sede. In sostanza, esse



LINE	# LOC	CODE	LINE
0002	0000		'ECO'
0003	0000		PROGRAMMA PER RICHIAMARE STRUTTURA SUONO ED
			ESEGUIRE ESP TEST.
0004	0000		L'UTENTE DEVE INDOVINARE UNA STRUTTURA DI LED
			ACCESI ED
0005	0000		I SUONI ASSOCIATI. LA COMBINAZIONE SUONO/LUCE PUO'
			ESSERE
0006	0000		VISUALIZZATA IN MODO CHE L'UTENTE PUO' RICORDARLA E
0007	0000		RICARICARLA CORRETTAMENTE.
0008	0000		FUNZIONAMENTO DEL PROGRAMMA:
0009	0000		L'INDIRIZZO DI PARTENZA E \$ 200
0010	0000		LA RIGA IN BASSO DI LED E' UN INDICATORE
0011	0000		DELLO STATO DEL PROGRAMMA: QUELLO PIU' A SINISTRA
0012	0000		(# 10) INDICA CHE IL PROGRAMMA ATTENDE
0013	0000		DALL'UTENTE LA LUNGHEZZA
0014	0000		DELLA SEQUENZA DA INDOVINARE
0015	0000		IL SECONDO LED DA SINISTRA (# 11) INDICA
0016	0000		CHE IL PROGRAMMA ATTENDE UN TENTATIVO (1-9),
0017	0000		IL COMANDO DI RESTART DEL GIOCO (0), OPPURE IL
0018	0000		COMANDO DI ESECUZIONE DELLA SEQUENZA (A-F).
0019	0000		I TASTI 1-9 SONO ASSOCIATI AI LED
0020	0000		1-9.
0021	0000		L'OSSERVAZIONE DELLA SEQUENZA A META' TENTATIVO
0022	0000		CANCELLERA' TUTTI I PRECEDENTI TENTATIVI
0023	0000		(RESET DI GESNO ED ERRS A 0).
0024	0000		DOPO UNA VITTORIA, SI HA IL RESTART DEL PROGRAMMA.
0025	0000		
0026	0000		COLLEGAMENTI:
0027	0000		GETKEY = \$ 100
0028	0000		
0029	0000		MEMORIZZAZIONE DELLE VARIABILI:
0030	0000	DIGITS = \$ 00	NUMERO DI DIGIT IN SEQUENZA
0031	0000	GESNO = \$ 01	NUMERO DEL TENTATIVO
			CORRENTE.
0032	0000		(A CHE PUNTO DELLA SERIE SI
			TROVA L'UTENTE)
0033	0000	ERRS = \$ 02	NUMERO DI ERRORI FATTI NELLA
0034	0000		SEQUENZA CORRENTE DI
			TENTATIVI.
0035	0000	DUR = \$ 03	MEMORIZZAZIONE
			TEMPORANEA DELLA DURATA
			DELLE NOTE.
0036	0000	FREQ = \$ 04	MEMORIZZAZIONE
			TEMPORANEA DELLA
			FREQUENZA DELLE NOTE.
0037	0000	TEMP = \$ 05	MEMORIZZAZIONE
			TEMPORANEA DEL REGISTRO X.
0038	0000	TABLE = \$ 06	MEMORIZZAZIONE DELLA
			SEQUENZA
0039	0000	RND = \$ 0F	SCRATCHPAD PER IL
			GENERATORE DI # CASUALI
0040	0000		INDIRIZZI DEL 6522 VIA # 1:
0041	0000	PORT1A = \$ A001	
0042	0000	DDR1A = \$ A003	
0043	0000	PORT1B = \$ A000	
0044	0000	DDR1B = \$ A002	
0045	0000	T1CL = \$ A004	
0046	0000		INDIRIZZI DEL 6522 VIA # 3:
0047	0000	PORT3B = \$ AC00	
0048	0000	DDR3B = \$ AC02	

Figura 8.5: Diagramma di flusso dell'Eco



0097	0259					; DI SUONI IN SEQUENZA.
0098	0259					
0099	0259	A2 00	SHOW	LDX # 0		
0100	025B	86 01		STX GESNO		; AZZERA TUTTI I TENTATIVI
						; CORRENTI.
0101	025D	86 02		STX ERRS		; AZZERA GLI ERRORI CORRENTI.
0102	025F	B5 06	SHOWLP	LDA TABLE,X		; ACCETTA L'X-ESIMO INGRESSO
						; NELLA TABELLA DELLA SERIE.
0103	0261	86 05		STX TEMP		; SALVA X
0104	0263	20 CF 02		JSR LIGHT		; ACCENDI LED # (TABLE (X))
0105	0266	20 FA 02		JSR PLAY		; ESEGUI TONO # (TABLE (X))
0106	0269	A0 FF		LDY # \$ FF		; IMPOSTA IL CONTATORE DEL
						; CICLO PER IL RITARDO
0107	026B	66 03	DELAY	ROR DUR		; CONSUMA TEMPO
0108	026D	26 03		ROL DUR		
0109	026F	88		DEY		; CONTO ALLA ROVESCIA ...
0110	0270	D0 F9		BNE DELAY		; SE NON FATTO, RICICLA ANCORA.
0111	0272	A6 05		LDX TEMP		; RIMEMORIZZA X
0112	0274	E8		INX		; INCREMENTA L'INDICE PER
						; MOSTRARE IL SUCCESSIVO
0113	0275	E4 00		CPX DIGITS		; MOSTRATI TUTTI I DIGIT?
0114	0277	D0 E6		BNE SHOWLP		; SE NO, VISUALIZZAZIONE
						; SUCCESSIVA.
0115	0279	F0 C9		BEQ KEY		; FATTO, ACCETTA L'INGRESSO
						; SUCCESSIVO.
0116	027B					
0117	027B					; ROUTINE PER VALUTARE I TENTATIVI DEL GIOCATORE.
0118	027B					
0119	027B	A6 01	EVAL	LDX GESNO		; ACCETTA IL NUMERO DI
						; TENTATIVI.
0120	027D	D5 06		CMP TABLE,X		; TENTATIVO = DIGIT
						; CORRISPONDENTE?
0121	027F	F0 0D		BEQ CORECT		; SE SI, VISUALIZZA AL
						; GIOCATORE.
0122	0281	E6 02	WRONG	INC ERRS		; TENTATIVO ERRATO, UN ALTRO
						; ERRORE.
0123	0283	A9 80		LDA # \$ 80		; DURATA DEL SUONO BASSO PER
						; INDICARE
0124	0285	85 03		STA DUR		; UN TENTATIVO ERRATO.
0125	0287	A9 FF		LDA # \$ FF		; COSTANTE DI FREQUENZA.
0126	0289	20 04 03		JSR PLYTON		; ESEGUI IL SUONO
0127	028C	F0 06		BEQ ENDCHK		; CONTROLLO DI FINE GIOCO
0128	028E	20 CF 02	CORECT	JSR LIGHT		; CONVALIDA IL TENTATIVO
						; CORRETTO.
0129	0291	20 FA 02		JSR PLAY		
0130	0294	E6 01	ENDCHK	INC GESNO		; CONSIDERATO UN ALTRO
						; TENTATIVO.
0131	0296	A5 00		LDA DIGITS		
0132	0298	C5 01		CMP GESNO		; INDOVINATI TUTTI I DIGIT?
0133	029A	D0 AB		BNE KEY		; SE NO, ACCETTA IL SUCCESSIVO.
0134	029C	A5 02		LDA ERRS		; ACCETTA IL NUMERO DI ERRORI.
0135	029E	C9 00		CMP # 0		; NESSUN ERRORE?
0136	02A0	F0 15		BEQ WIN		; SE NO, IL GIOCATORE VINCE.
0137	02A2	20 CF 02	LOSE	JSR LIGHT		; MOSTRA IL NUMERO DI ERRORI.
0138	02A5	A9 09		LDA # 9		; ESEGUI OTTO SUONI
						; DECRESCENTI
0139	02A7	48	LOSELP	PHA		
0140	02A8	20 FA 02		JSR PLAY		
0141	02AB	68		PLA		
0142	02AC	38		SEC		

Figura 8.5: Diagramma di flusso dell'Eco (continua)

0143	02AD	E9 01		SBC # 1	
0144	02AF	D0 F6		BNE LOSELP	
0145	02B1	85 01		STA GESNO	: AZZERA LE VARIABILI
0146	02B3	85 02		STA ERRS	
0147	02B5	F0 8D		BEQ KEY	: ACCETTA IL SUCCESSIVO
					: TENTATIVO DELLA SEQUENZA
0148	02B7	A9 FF	WIN	LDA # \$ FF	: ACCENDI TUTTI I LED PER UNA
					: VITTORIA
0149	02B9	8D 01 A0		STA PORT1A	
0150	02BC	8D 00 A0		STA PORT1B	
0151	02BF	A9 01		LBA # 1	: ESEGUI OTTO SUONI CRESCENTI
0152	02C1	48	WINLP	PHA	
0153	02C2	20 FA 02		JSR PLAY	
0154	02C5	68		PLA	
0155	02C6	18		CLC	
0156	02C7	69 01		ADC # 01	
0157	02C9	C9 0A		CMP # 10	
0158	02CB	D0 F4		BNE WINLP	
0159	02CD	F0 84		BEQ STRTJP	: ESEGUI UN SALTO DOPPIO PER IL
					: RESTART
0160	02CF				
0161	02CF				: ROUTINE PER ACCENDERE L'N-ESIMO LED DOVE N È
0162	02CF				: IL NUMERO TRASFERITO COME PARAMETRO
0163	02CF				: NELL'ACCUMULATORE.
0164	02CF				
0165	02CF	48	LIGHT	PHA	: SALVA A
0166	02D0	A8		TAY	: USA A COME CONTATORE IN Y
0167	02D1	A9 00		LDA # 0	: AZZERA A PER LO SCORRIMENTO
					: DI BIT
0168	02D3	8D 00 A0		STA PORT1B	: AZZERA I LED HI
0169	02D6	38		SEC	: GENERA HI BIT PER LO
					: SCORRIMENTO A SINISTRA
0170	02D7	2A	LTSHT	ROL A	: MUOVI HI BIT A SINISTRA
0171	02D8	88		DEY	: DECREMENTA CONTATORE
0172	02D9	D0 FC		BNE LTSHT	: SCORRIMENTO ESEGUITO?
0173	02DB	8D 01 A0		STA PORT1A	: MEMORIZZA LA STRUTTURA
					: CORRETTA
0174	02DE	90 05		BCC LTCC	: BIT 9 NON HI, FATTO
0175	02E0	A9 01		LDA #1	
0176	02E2	8D 00 A0		STA PORT1B	: ACCENDI IL LED 9.
0177	02E5	68	LTCC	PLA	: RIMEMORIZZA A
0178	02E6	60		RTS	: FATTO.
0179	02E7				
0180	02E7				: GENERATORE DI NUMERI CASUALI: SI RITORNA CON UN
0181	02E7				: NUOVO NUMERO CASUALE IN A.
0182	02E7				
0183	02E7	38			
			RANDOMSEC		
0184	02E8	A5 10		LDA RND+ 1	
0185	02EA	65 13		ADC RND+ 4	
0186	02EC	65 14		ADC RND+ 5	
0187	02EE	85 0F		STA RND	
0188	02F0	A2 04		LDX # 4	
0189	02F2	B5 0F	RNDLP	LDA RND,X	
0190	02F4	95 10		STA RND + 1,X	
0191	02F6	CA		DEX	
0192	02F7	10 F9		BPL RNDLP	
0193	02F9	60		RTS	
0194	02FA				
0195	02FA				: ROUTINE PER L'ESECUZIONE DEL SUONO IL CUI NUMERO È

Figura 8.5: Diagramma di flusso dell'Eco (continua)

```

0196 02FA      : STATO TRASFERITO ATTRAVERSO L'ACCUMULATORE. SE SI
      : ENTRA PLYTON,
0197 02FA      : SI ESEGUIRA' IL SUONO LA CUI LUNGHEZZA È IN DUR,
0198 02FA      : FREQUENZA NELL'ACCUMULATORE.
0199 02FA      :
0200 02FA A8    PLAY    TAY      : USA # SUONO COME INDICE ...
0201 02FB 88      DEY      : DECREMENTA PER
      : CONFRONTARE LE TABELLE
0202 02FC B9 27 0D    LDA DURTAB,Y : ACCETTA LA DURATA DEL #
      : SUONO N.
0203 02FF 85 03      STA DUR      : SALVALA.
0204 0301 B9 1E 03    LDA NOTAB,Y : ACCETTA LA COSTANTE DI
      : FREQUENZA DEL # SUONO N
0205 0304 B5 04    PLYTON STA FREQ : SALVALA.
0206 0306 A9 00      LDA # 0      : PONI BASSA LA PORTA
      : DELL'ALTOPARLANTE
0207 0308 8D 00 AC    STA PORT3B
0208 030B A6 03      LDX DUR      : ACCETTA LA DURATA IN # DI 1/2
      : CICLI
0209 030D A4 04      LDY FREQ     : ACCETTA LA FREQUENZA
0210 030F 88      FL1    DEY      : CONTO ALLA ROVESCIA DEL
      : RITARDO
0211 0310 18      CLC            : CONSUMA TEMPO
0212 0311 90 00      BCC # + 2
0213 0313 D0 FA      BNE FL1      : CICLO DEL RITARDO
0214 0315 49 FF      EOR # $ FF   : COMPLEMENTA LA PORTA
0215 0317 8D 00 AC    STA PORT3B
0216 031A CA      DEX            : CONTO ALLA ROVESCIA DEL
      : RITARDO...
0217 031B D0 F0      BNE FL2      : CICLO FINO ALL'ESECUZIONE
      : DELLA NOTA.
0218 031D 60      RST            : FATTO.
0219 031E
0220 031E      : TABELLA PER LE FREQUENZE DELLE NOTE.
0221 031E
0222 031E C9      NOTAB . BYTE $ C9, $ BE, $ A9, $ 96, $ 8E, $ 7E, $ 70, $ 64, $ 5E
0222 031F BE
0222 0320 A9
0222 0321 96
0222 0322 8E
0222 0323 7E
0222 0324 70
0222 0325 64
0222 0326 5E
0223 0327
0224 0327      : TABELLA PER LE DURATE DELLE NOTE.
0225 0327
0226 0327 6B      DURTAB . BYTE $ 6B, $ 72, $ 80, $ 8F, $ 94, $ AA, $ BF, $ D7, $ E4
0226 0328 72
0226 0329 80
0226 032A 8F
0226 032B 94
0226 032C AA
0226 032D BF
0226 032E D7
0226 032F E4
0227 0330      . END

```

Figura 8.5: Diagramma di flusso dell'Eco (continua)

SYMBOL TABLE							
SYMBOL	VALUE						
CORECT	028E	DDR1A	A003	DDR1B	A002	DDR3B	AC02
DELAY	026B	DIGITS	0000	DIGKEY	0220	DUR	0003
DURTAB	0327	ENDCHK	0294	ERRS	0002	EVAL	027B
FILL	022F	FL1	030F	FL2	030D	FREQ	0004
GESNO	0001	GETKEY	0100	KEY	0244	LIGHT	02CF
LOSE	02A2	LOSELP	02A7	LTCC	02E5	LISHFT	02D7
NOTAB	031E	PLAY	02FA	PLYTON	0304	PORT1A	A001
PORT1B	A000	PORT3B	AC00	RANDOM	02E7	RND	000F
RNDLP	02F2	SHOW	0259	SHOWLP	025F	START	0200
STRTJP	0253	T1CL	A004	TABLE	0006	TEMP	0005
WIN	02B7	WINLP	02C1	WRONG	0281		
END OF ASSEMBLY							

Figura 8.5: Diagramma di flusso dell'Eco (continua)

forniscono le costanti di ritardo richieste per implementare una nota di frequenza appropriata e per eseguirla per il periodo di tempo appropriato. Notate che è possibile modificare la difficoltà del gioco aumentando o diminuendo la durata di esecuzione della nota. Chiaramente, riducendo la durata si rende il gioco più difficile. Aumentando la durata lo si rende normalmente più facile, fino ad un certo punto però. Si consiglia vivamente di provare queste variazioni.

Le variazioni principali impiegate dal programma sono le seguenti:

DIGITS contiene il numero di digit della sequenza da riconoscere.

GESNO indica il numero del tentativo in corso, cioè quale delle note della serie l'utente sta cercando di riconoscere.

ERRS indica il numero di errori eseguiti dal giocatore.

TABLE è la tabella contenente la sequenza da riconoscere.

Qualche altra locazione di memoria è riservata ai parametri passanti alle subroutine o come memoria scratch-pad. Esse verranno descritte all'interno del contesto delle routine associate.

Come al solito, il programma inizia settando il registro di direzione dati della Porta 1A, della Porta 1B e della Porta 3B ad una configurazione d'uscita:

```
START      LDA #$FF
           STA DDR1A
           STA DDR1B
           STA DDR3B
```

Successivamente, i LED accesi sulla scheda vengono spenti:

```
LDA #0
STA PORT1A
```

e le due variabili, ERRS e GESNO, sono poste a zero:

STA ERRS  
STA GESNO

Viene attivato il generatore di numeri casuali ottenendo un "seme" che viene memorizzato alle locazioni RND+1 ed RND+4:

LDA TICL	Leggi il contatore timer
STA RND+1	
STA RND+4	

Il gioco è ora pronto. Occorre accendere il LED 10 per indicare al giocatore che il gioco è pronto:

LDA #%010	Struttura per il LED 10
STA PORT 1B	Specifica la lunghezza

Viene quindi esplorata la tastiera per ottenere l'ingresso del giocatore con l'impiego della solita subroutine GETKEY (descritta al Capitolo 1):

DIGKEY JSR GETKEY

Si confronta l'ingresso con il valore "0":

CMP#0	
BEQ DIGKEY	Se = 0, accettane un altro

Se l'ingresso era "0", il programma attende un'altra pressione di tasto. Altrimenti si esegue il confronto con il valore 10:

CMP #10	Sequenza più lunga di 9
BPL DIGKEY	

Se la lunghezza della sequenza è maggiore di 9, si ha un altro rigetto. L'accettazione dei soli ingressi validi, impiegando un filtro è nota come "testing di ragionevolezza".

Se tutto va bene, la lunghezza della sequenza da riconoscere viene memorizzata nella locazione di memoria DIGITS:

STA DIGITS	Lunghezza della sequenza
------------	--------------------------

Viene quindi calcolato un puntatore corrente e memorizzato alla loca-

zione di memoria TEMP. Esso è uguale alla lunghezza precedente meno 1:

	TAX	Impiega X per il calcolo
	DEX	Decrementa
FILL	STX TEMP	

Viene quindi chiamata la subroutine RANDOM per fornire un primo numero casuale:

JSR RANDOM

Il puntatore di posizione nella serie di note che stanno per essere generate viene recuperato da TEMP e memorizzato nel registro indice X, anticipando la memorizzazione del nuovo numero casuale in TABLE:

LDX TEMP

Il valore del numero casuale contenuto nell'accumulatore viene successivamente convertito in un valore decimale compreso tra 0 e 9. Questo processo può essere eseguito in vari modi. In questo caso si sfrutta lo speciale modo decimale disponibile sul 6502. Il modo decimale viene impostato specificando:

SED	Imposta il modo decimale
-----	--------------------------

Si noti che il flag di carry deve essere azzerato prima di un'addizione:

CLC	Azzera il carry
-----	-----------------

Il trucco impiegato in questo caso è di sommare "0" al numero casuale contenuto nell'accumulatore. Ne risulta che nella parte destra di A c'è sicuramente un digit compreso tra 0 e 9, poichè stiamo operando in modo decimale. Naturalmente potrebbe essere sommato ad A qualsiasi altro numero per rendere "decimali" i suoi contenuti; comunque questo potrebbe cambiare la distribuzione dei numeri casuali e nella serie non potrebbero mai apparire i numeri 0, 1 e 2. Una volta eseguita questa conversione, il modo decimale viene semplicemente disabilitato:

ADC #0	Somma "0" in modo decimale
CLD	Disabilita il modo decimale

Questa è una potente prestazione del 6502 impiegata con grande vantaggio in questa occasione. Per assicurarci che il risultato rimasto in A sia un



numero decimale tra 0 e 9, il nibble superiore del byte viene rimosso mediante mascheratura:

AND #\$0F

Infine, un valore "0" non è consentito e deve essere ottenuto un nuovo numero se questo è il valore corrente dell'accumulatore:

BEQ FILL

**Esercizio 8-2:** *Potremmo evitare questo caso speciale di valore 0 sommando un valore oltre a "0" ad A precedente?*

Se questo non è il valore corrente dell'accumulatore, abbiamo un numero decimale tra 1 e 9 che è ragionevolmente casuale, che ora può essere memorizzato nella tabella. Ricordate che il registro indice X è stato precaricato con la posizione del numero corrente nella sequenza (recuperata dalla locazione di memoria TEMP). Esso può essere impiegato come indice:

STA TABLE, X	Memorizza il numero nella tabella
--------------	--------------------------------------

Il puntatore del numero viene quindi decrementato per anticipare l'iterazione successiva:

DEX

ed il ciclo viene ri-eseguito fino al riempimento della tabella di numeri casuali:

BPL FILL

Ora siamo pronti per eseguire il gioco. Verrà acceso il LED 11 per segnalare al giocatore che può introdurre un tentativo:

KEY	LDA #0
	STA PORT1A
	LDA #%0100
	STA PORT1B

Il tentativo del giocatore viene quindi letto dalla tastiera:

JSR GETKEY

Accetta il tentativo

Occorre controllare se è "0" oppure un valore alfabetico. Eseguiamo il test per "0":

	CMP #0	È 0?
STRTJP	BEQ START	Se sì, restart

Se era "0", il gioco viene ri-inizializzato e si salta alla locazione START. Se non era "0", dobbiamo controllare se si tratta di un carattere alfabetico:

	CMP #10	Numero < 10?
	BMI EVAL	Se sì, valuta il grado di correttezza

Se il valore d'ingresso del tasto premuto è maggiore di 10, si tratta di un tentativo che viene valutato mediante la routine EVAL. In caso contrario il programma esegue la routine SHOW per visualizzare la serie.

### *La routine SHOW*

Assumiamo in questo caso che sia stato premuto un tasto alfabetico. Il test BMI non è soddisfatto ed entriamo nella routine SHOW. Questa routine esegue l'accordo generato dal computer e genera l'accensione della sequenza corrispondente di LED. Inoltre, ogni volta che si accede a questa routine, viene ri-inizializzata la sequenza di tentativi e le variabili temporanee vengono resettate a 0:

SHOW	LDX #0	
	STX GESNO	
	STX ERRS	Reset di tutte le variabili

Si ottiene il primo ingresso alla tabella, viene acceso il LED corrispondente ed eseguita la nota corrispondente:

SHOWLP	LDA TABLE,X	Accetta l'X-esimo ingresso alla tabella
	STX TEMP	Salva X
	JSR LIGHT	Accendi LED # TABLE (X)
	JSR PLAY	Esegui la nota # TABLE (X)

Viene quindi implementato un ritardo tra le note impiegando Y come contatore del ciclo e due pseudo-istruzioni per aumentare il ritardo:

	LDY #\$FF	
DELAY	ROR DUR	Pseudo-istruzione
	ROL DUR	Pseudo-istruzione
	DEY	Conto alla rovescia
	BNE DELAY	Fine del ciclo di test

Ora siamo pronti per eseguire la stessa operazione per la nota successiva della tabella corrente. Il puntatore indice viene ri-memorizzato ed incrementato:

LDX TEMP	Ri-memorizza X
INX	Incrementalo

Esso viene quindi confrontato con il numero massimo di digit memorizzati nella tabella. Se è stato raggiunto il massimo, l'operazione di visualizzazione è completata e si ritorna indietro alla label KEY. In caso contrario viene eseguita la nota successiva e si ritorna indietro alla label SHOWLP:

CPX DIGITS	Mostrati tutti i digit?
BNE SHOWLP	
BEQ KEY	Fatto, accetta l'ingresso successivo

### *La routine EVAL*

Esaminiamo ora la routine che valuta il tentativo del giocatore. Si tratta della routine EVAL. Il valore dell'ingresso corrispondente in TABLE viene ottenuto e confrontato con l'ingresso del giocatore:

EVAL	LDX GESNO	Carica il numero di tentativo in X
	CMP TABLE,X	Confronta il tentativo con il numero
	BEQ CORECT	Se corretto, avverti il giocatore

Se c'è accordo si salta alla locazione CORECT; altrimenti il programma procede alla label WRONG. Esaminiamo questo caso. Se il tentativo è errato, viene registrato un ulteriore errore:

WRONG	INC ERRS
-------	----------

Viene eseguita una nota bassa:

LDA #\$80	
STA DUR	
LDA #\$FF	
JSR PLYTON	Esegui la nota

Quindi si verifica un salto alla locazione ENDCHK:

BEQ ENDCHK	Controlla se fine del gioco
------------	-----------------------------

**Esercizio 8-3:** *Esamine la precedente istruzione BEQ. Essa genererà sempre un salto alla label ENDCHK? (Suggerimento: determinate se il bit Z sarà uno oppure zero a questo punto).*

**Esercizio 8-4:** *Quali sono i vantaggi dell'impiego di BEQ (precedente) rispetto a JMP?*

Ora consideriamo cosa succede nel caso di un tentativo corretto. Se il tentativo è corretto si accende il LED corrispondente e viene eseguita la relativa nota. Entrambe le subroutine assumono che l'accumulatore contenga il numero specificato:

CORECT	JSR LIGHT	Accendi il LED
	JSR PLAY	Esegui la nota per conferma

Ora dobbiamo determinare se abbiamo raggiunto la fine di una sequenza oppure no e prendere i provvedimenti adeguati. Il numero di tentativi viene quindi incrementato e confrontato con la lunghezza massima dell'accordo memorizzato:

ENDCHK	INC GESNO	Altro tentativo
	LDA DIGITS	
	CMP GESNO	Indovinati tutti i digit?
	BNE KEY	Se no, prendi la chiusura di un tasto successiva

Se non è ancora finita, si verifica un salto indietro alla label KEY. Altrimenti, abbiamo raggiunto la fine di un gioco e dobbiamo segnalare una situazione di vittoria o di sconfitta. Per determinare questo viene controllato il numero di errori:

LDA ERRS	Accetta il numero di errori
CMP #0	Nessun errore?
BEQ WIN	Se no, il giocatore vince

Se viene riconosciuta una "vittoria", si verifica una ramificazione alla label WIN. Questo verrà descritto in seguito. Esaminiamo ora cosa succede nel caso di una sconfitta:

LOSE	JSR LIGHT	Mostra il numero di errori
------	-----------	----------------------------

Il numero di errori viene visualizzato accendendo il LED corrispondente. Ricordate che l'accumulatore era stato condizionato prima di entrare in questa routine e conteneva il valore di ERRS, cioè il numero di errori.

Successivamente viene eseguita una sequenza di otto note decrescenti. La sommità dello stack viene impiegata per contenere il numero rimanente di note da eseguire:

LOSELP	LDA #9	Esegui otto note decrescenti
	PHA	Salva A nello stack
	JSR PLAY	Esegui la nota
	PLA	Ri-memorizza A

Una volta che è stata eseguita una nota, viene decrementato di 1 il numero di note da eseguire e si esegue il test per "0":

SEC	Poni ad 1 il carry (per la sottrazione)
SBC #1	Sottrai uno
BNE LOSELP	

**Esercizio 8-5:** *Notate come la sommità dello stack è stata impiegata come una locazione di scratch temporaneo. Siete in grado di suggerire un modo alternativo per ottenere lo stesso risultato senza impiegare lo stack?*

**Esercizio 8-6:** *Discutete i meriti relativi all'impiego dello stack rispetto all'impiego di altre tecniche per fornire locazioni di lavoro temporanee al programma. Esistono dei pericoli potenziali nell'impiego dello stack?*

Vengono eseguite otto note successive. Quindi si ha il reset a "0" delle due variabili di lavoro GESNO ed ERRS. Si verifica successivamente una ramificazione indietro all'inizio del programma:

STA GESNO	Azzera le variabili
STA ERRS	
BEQ KEY	Accetta la successiva sequenza di tentativo

Esaminiamo ora cosa succede in una situazione di vittoria. Tutti i LED sulla Scheda Giochi vengono accesi contemporaneamente:

WIN	LDA #\$FF	È una vittoria: accendi tutti i LED
	STA PORT1A	
	STA PORT1B	

Successivamente viene eseguita una sequenza di otto note crescenti. Il numero della nota viene memorizzato nell'accumulatore e verrà impiegato come indice dalla subroutine *PLAY* per generare una nota appropriata. Come prima, viene impiegata la sommità dello stack per fornire la memoria di lavoro:

WINLP	LDA #1	A sarà incrementato fino a 9
	PHA	Salva A sullo stack
	JSR PLAY	
	PLA	

Viene quindi incrementato di uno il numero di note che sono state eseguite e si esegue il confronto con il valore massimo 9:

CLC	Azzera carry per l'addizione
ADC #01	
CMP #10	

Finchè non viene raggiunto il massimo pari a 9, si verifica una ramificazione indietro alla label *WINLP*:

BNE WINLP

Altrimenti, si inizia un nuovo gioco:

BEQ STRTJP	Doppio salto per restart
------------	--------------------------

Questo completa la descrizione del programma principale. Questo programma impiega tre subroutine, che verranno ora descritte.

## Le Subroutine

### *Subroutine LIGHT*

Questa subroutine assume che l'accumulatore contenga il numero del LED da accendere. La subroutine accende il LED appropriato sulla Scheda Giochi. Si otterrà questo risultato scrivendo un "1" nella posizione corretta dell'accumulatore e quindi inviando il contenuto alla porta d'uscita corretta. Si può usare sia la Porta 1A (per i LED da 1 a 8), sia la Porta 1B (per il LED 9). Il bit "1" viene scritto nella posizione

corretta dell'accumulatore mediante l'esecuzione di una sequenza di scorrimenti. Il numero di scorrimenti è uguale alla posizione del LED da accendere. Come contatore di scorrimenti viene impiegato il registro indice Y. Il numero del LED da accendere viene salvato nello stack all'inizio della subroutine e sarà ripristinato all'uscita. Notate che questo è un modo classico per preservare i contenuti di un registro essenziale durante l'esecuzione della subroutine, cosicchè i contenuti dell'accumulatore saranno invariati all'uscita della subroutine. In caso contrario il programma chiamante dovrebbe aver preservato esplicitamente i contenuti dell'accumulatore prima della chiamata della subroutine LIGHT. Quindi dovrebbe averli ricaricati nell'accumulatore prima dell'impiego di un'altra routine, quale la PLAY. Poichè le subroutine LIGHT e PLAY sono normalmente impiegate in sequenza, è molto più efficiente affidare alla responsabilità della subroutine il salvataggio dei contenuti dell'accumulatore. Eseguiamolo:

LIGHT	PHA	Salva A
-------	-----	---------

Viene quindi predisposto il contatore di scorrimenti:

TAY	Impiega Y come contatore di scorrimenti
-----	---

e l'accumulatore viene inizializzato a "0":

LDA #0	Azzera A
--------	----------

Il LED 9 viene spento nel caso fosse acceso:

STA PORT1B

Viene quindi implementato il ciclo di scorrimento. Il bit carry viene inizialmente posto ad "1" e sarà fatto scorrere a sinistra nell'accumulatore il numero di volte necessario:

LTSHFT	SEC	Poni ad "1" il carry
	ROL A	
	DEY	
	BNE LTSHFT	

La struttura di bit corretta è ora contenuta nell'accumulatore e viene visualizzata sulla Scheda Giochi:

STA PORT1A

Comunque, si può verificare un'eccezione: se il LED 9 è stato specificato, i contenuti dell'accumulatore a questo punto sono "0" ma il bit carry è stato posto ad "1" dall'ultimo scorrimento. Questo caso può essere rivelato mediante:

BCC LTCC                      È posto ad "1" il bit 9?

Se ci si trova in questa situazione, l'accumulatore deve essere posto al valore "00000001", che viene inviato alla Porta d'uscita 1B:

LDA #1  
STA PORT1B                      Accendi il LED 9

Infine usciamo dalla routine senza dimenticare di ripristinare l'accumulatore dallo stack dove era stato salvato:

LTCC                      PLA                      Rimemorizza A  
                                 RTS

**Esercizio 8-7:** *Elencate i registri distrutti o alterati da questa subroutine ogni volta che essa viene eseguita.*

**Esercizio 8-8:** *Assumete che il registro Y debba rimanere invariato al ritorno da questa subroutine. Quali sono le variazioni necessarie al programma, se esistono?*

### *Subroutine RANDOM*

Questa subroutine genera un nuovo numero casuale e rinvia il suo valore in A. Il suo funzionamento è già stato descritto al Capitolo 4.

### *Subroutine PLAY*

Normalmente questa subroutine eseguirà la nota corrispondente al numero contenuto nell'accumulatore. Opzionalmente essa può essere caricata alla locazione di memoria PLYTON e quindi viene eseguita la nota corrispondente alla frequenza impostata dall'accumulatore e corrispondente alla lunghezza specificata dai contenuti della locazione di memoria DUR. Esaminiamo questa subroutine.

Il registro indice Y viene impiegato come indice delle due tabelle richieste per determinare la durata e la frequenza della nota. In questo gioco possono essere eseguite fino a 9 note, corrispondenti ai LED ed ai tasti da 1 a 9. Inizialmente viene condizionato il registro indice Y:



PLAY	TAY	Usa la # nota come indice
	DEY	Decrementa al valore interno

Notate che il registro indice deve essere decrementato di uno. Questo perchè il tasto 1 corrisponde al numero d'ingresso 0 nella tabella, e così via. La durata e la frequenza vengono ottenute dalle tabelle DURTAB e NOTAB impiegando un modo di indirizzamento indicizzato. Esse vengono memorizzate rispettivamente alle locazioni DUR e FREQ:

	LDA DURTAB,Y	Accetta la durata
	STA DUR	Salvala
	LDA NOTAB,Y	Accetta la frequenza
PLYTON	STA FREQ	Salvala

L'altoparlante viene quindi posto off:

LDA #0	
STA PORT3B	Setta la Porta 3B per l'altoparlante

Verranno ora implementati due cicli. Un ciclo interno impiegherà il registro Y come contatore di ritardo per implementare la frequenza corretta della nota.

Il registro X verrà impiegato nel ciclo esterno e genererà la nota per la durata corretta.

Condizioniamo i due registri contatori:

	LDX DUR	Accetta la durata in # di 1/2 cicli
FL2	LDY FREQ	Accetta la frequenza

Successivamente implementiamo il ritardo del ciclo interno:

FL1	DEY	
	CLC	Consuma tempo
	BCC * + 2	
	BNE FL1	Ciclo di ritardo

Notate che le due istruzioni di pausa sono state poste all'interno del ciclo per generare un ritardo più lungo. Alla fine di questo ciclo di ritardo interno, i contenuti della porta d'uscita connessa all'altoparlante sono complementati in modo da generare un'onda quadra.

EOR #\$FF	Complementa la porta
-----------	----------------------

Notate che, ancora una volta, EOR #\$FF viene impiegata per complementare i contenuti di un registro.

### STA PORT3B

Viene quindi completato il ciclo esterno:

```
DEX  
BNE FL2      Ciclo esterno  
RTS
```

## SOMMARIO

Questo programma dimostra quanto sia semplice implementare da tastiera un gioco elettronico che esegua dei suoni come ingresso/uscita e sfidi i giocatori.

**Esercizio 8-9:** *Le costanti di durata e di frequenza delle nove note sono mostrate in Fig. 8.6. Quali sono le frequenze effettive generate dal programma?*

NOTA	COSTANTE DI FREQUENZA	COSTANTE DI DURATA
1	C9	68
2	BE	72
3	A9	80
4	96	8F
5	8E	94
6	7E	AA
7	70	BF
8	64	D7
9	5E	E4

Figura 8.6: Costanti di frequenza e durata

## CAPITOLO 9

# MINDBENDER

### LE REGOLE

Questo gioco è ispirato da quello commerciale MasterMind (Marchio registrato dal produttore, Invicta Plastics, Ltd.). In questo gioco uno o più giocatori competono con il computer (e tra di loro). Il computer genera una sequenza di digit (cifre) - per esempio una sequenza di cinque digit tra "0" e "9" - ed il giocatore tenta di indovinare la sequenza dei cinque numeri in ordine corretto. Il computer risponde al giocatore comunicandogli il numero di digit indovinati e la loro posizione nella sequenza numerica.

I LED da 1 a 9 sulla Scheda Giochi vengono impiegati per visualizzare il responso del computer. Un LED lampeggiante viene impiegato per indicare che il tentativo del giocatore contiene un digit corretto nella posizione esatta della sequenza. Un LED sempre acceso viene impiegato per indicare un digit indovinato ma non in sequenza. Diversi giocatori possono confrontare tra loro la propria abilità. Per un prefissato livello di difficoltà - diciamo per indovinare una sequenza di sette digit - risulta vincitore quel giocatore che riesce ad indovinare correttamente la sequenza di numeri con il minor numero di tentativi.

Il gioco può anche essere eseguito con handicap (svantaggio) in cui un giocatore deve indovinare una sequenza di  $n$  digit, mentre l'avversario deve indovinare una sequenza di soli  $n-1$  digit. Questo è un notevole svantaggio, poichè l'aumento di uno del livello di difficoltà è abbastanza significativo.

### UNA GIOCATA TIPICA

Per eseguire questo gioco viene impiegata sia la retroazione sonora che visiva.

## La retroazione sonora

Ogni volta che un giocatore ha introdotto la sua sequenza di tentativi, il computer risponde eseguendo un suono specifico. Un suono basso indica un tentativo non corretto; uno alto che la sequenza è corretta.

## La retroazione visiva

All'inizio di ciascun gioco, è acceso il LED # 10, che indica la richiesta della lunghezza della sequenza da indovinare. Questo è mostrato in Fig. 9.1. Il giocatore quindi specifica la lunghezza della sequenza con un numero da 1 a 9. Qualsiasi altro ingresso verrà ignorato.

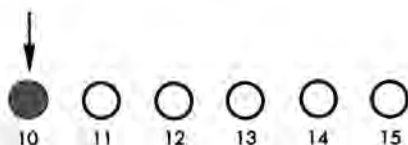


Figura 9.1: Carica la lunghezza della sequenza

Non appena la lunghezza è stata specificata, per esempio supponiamo sia stata selezionata la lunghezza "2", si accende il LED # 11. Questo significa "introduci il tuo tentativo". (Vedere Fig. 9.2). A questo punto il giocatore carica la sua sequenza di tentativo di due digit. Eseguiamo ora un gioco.

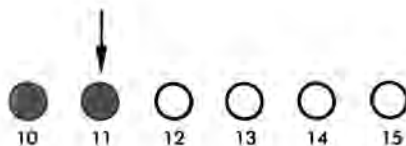


Figura 9.2: Caricate il vostro tentativo

Il giocatore batte la sequenza "1,2". Viene eseguito un suono basso, i LED 10 ed 11 vengono spenti per un breve periodo ma non succede nient'altro. La situazione è indicata in Fig. 9.3. Poiché i LED da 1 a 9 non sono accesi, non esiste alcuna cifra (digit) corretta nel tentativo. I digit "1" e "2" devono essere eliminati. Eseguiamo un altro tentativo.

Battiamo "3,4". Viene eseguito un suono basso ma questa volta il

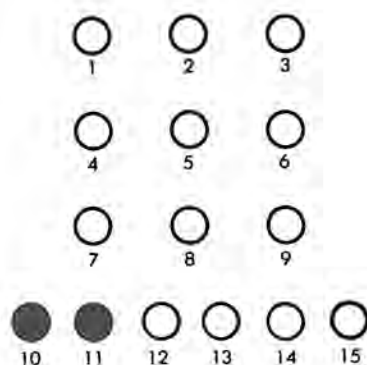


Figura 9.3: Il giocatore carica un tentativo errato

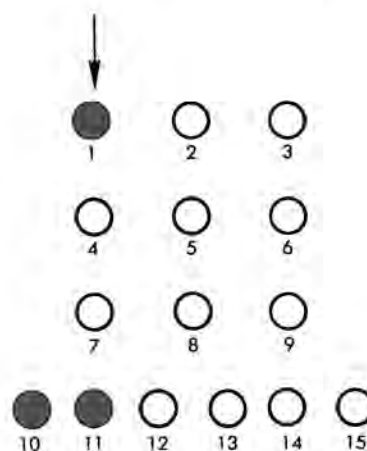


Figura 9.4: Un digit corretto nella posizione corretta

LED # 1 rimane acceso in modo continuo, come indicato in Fig. 9.4. Da questo deduciamo che il “3” oppure il “4” è una cifra corretta ma si trova nella posizione sbagliata. Inversamente, la sequenza “4,3” deve avere un digit corretto nella posizione corretta. Per sicurezza eseguiamo un test. Battiamo “4,3”. Viene eseguito un suono basso, indicante che la sequenza non è corretta, ma questa volta il LED # 1 è acceso e lampeggia. Questo dimostra che il nostro ragionamento è corretto, e procediamo.

Ora proviamo “4,5”. Viene eseguito un suono di altezza elevata ed i

LED 1 e 2 si accendono brevemente, indicando che le cifre sono state indovinate correttamente e che abbiamo vinto il nostro primo gioco.

Alla fine del gioco, la situazione ritorna a quella iniziale, come indicato in Fig. 9.1. Notate che battendo un valore diverso da 1 a 9 al posto di un tentativo si ha il restart del gioco.

Esiste una peculiarità del gioco: se il numero del tentativo contiene due digit identici ed il giocatore carica questo digit particolare in una delle due locazioni corrette, il computer risponderà indicando che si ha un digit giusto nella posizione giusta ed un digit giusto nella posizione sbagliata!

## L'ALGORITMO

La Fig. 9.5 riporta il diagramma di flusso del gioco Mindbender. Per eseguire il lampeggio dei LED vengono impiegati degli interrupt. Gli interrupt saranno generati automaticamente dal timer ad intervallo programmabile del VIA # 1 ad intervalli di circa 1/15 di secondo.

Con riferimento alla Fig. 9.5, vengono preliminarmente inizializzati tutti i registri e le locazioni di memoria richieste. Successivamente (blocco 2 del diagramma di flusso), viene letta dalla tastiera la lunghezza della sequenza da indovinare. Viene impiegato il test di ragionevolezza per "filtrare" da 1 a 9 l'ingresso del giocatore.

Successivamente deve essere generata una sequenza di numeri casuali. Nel blocco 3 del diagramma di flusso, viene generata una sequenza di numeri casuali, che viene memorizzata in una tabella di digit, con inizio all'indirizzo DIG0.

Nel blocco 5, la sequenza di numeri del computer viene confrontata - un numero alla volta - con il tentativo del giocatore. L'algoritmo accetta un digit dalla sequenza del computer e lo confronta con ogni digit della sequenza del giocatore. Come si è già visto, questo può generare l'accensione di due LED, se ci sono due o più digit identici nel numero da indovinare e se il giocatore ha specificato un solo digit. Un digit può trovarsi nella giusta posizione, mentre altri, pur essendo corretti possono trovarsi nella posizione errata.

Notate che, alternativamente, si potrebbe impiegare un altro algoritmo di confronto in cui ogni digit della sequenza del giocatore viene confrontato di volta in volta con ciascun digit della sequenza del computer.

Una volta che sono stati confrontati tutti i digit, viene visualizzato il punteggio risultante sui LED (blocco 6). Infine, si esegue un test per verificare se si tratta di una situazione vincente (blocco 7) e viene generato il suono opportuno (blocco 8).

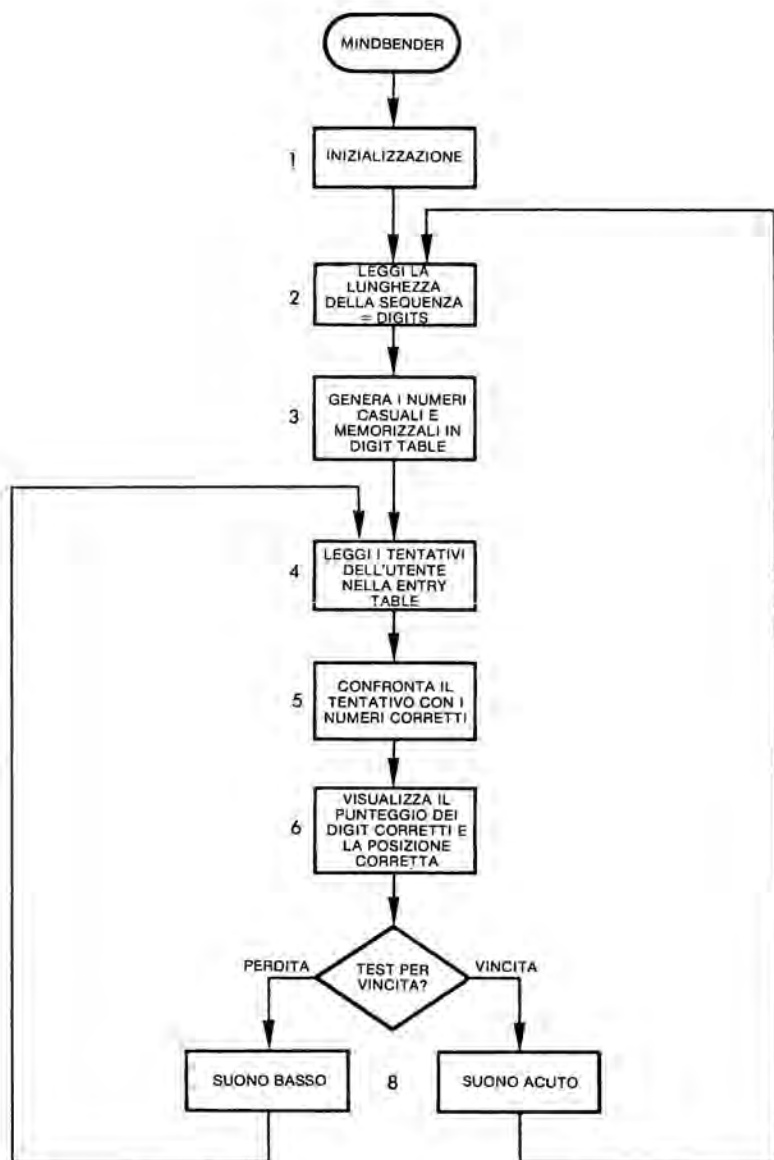


Figura 9.5: Diagramma di flusso del Mindbender

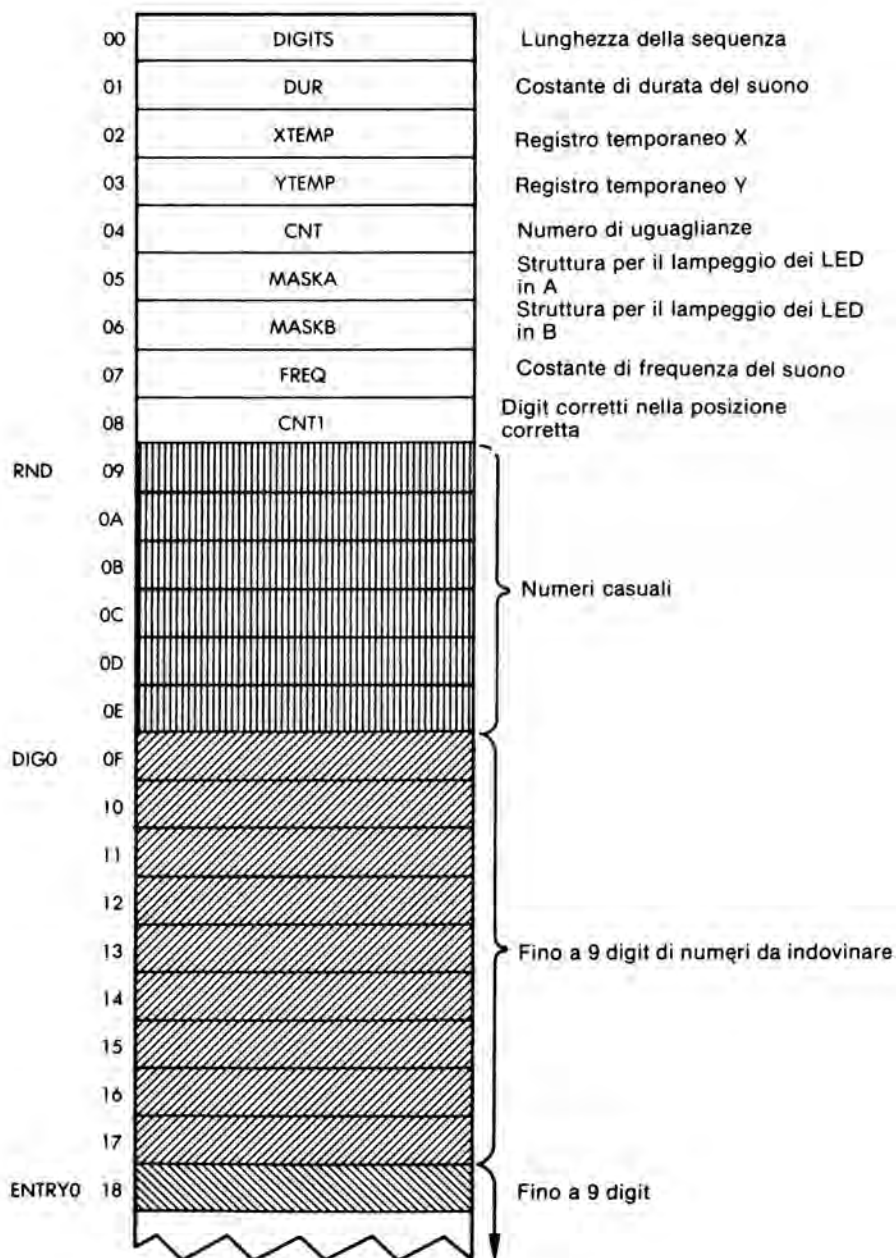


Figura 9.6: Mappa della memoria bassa



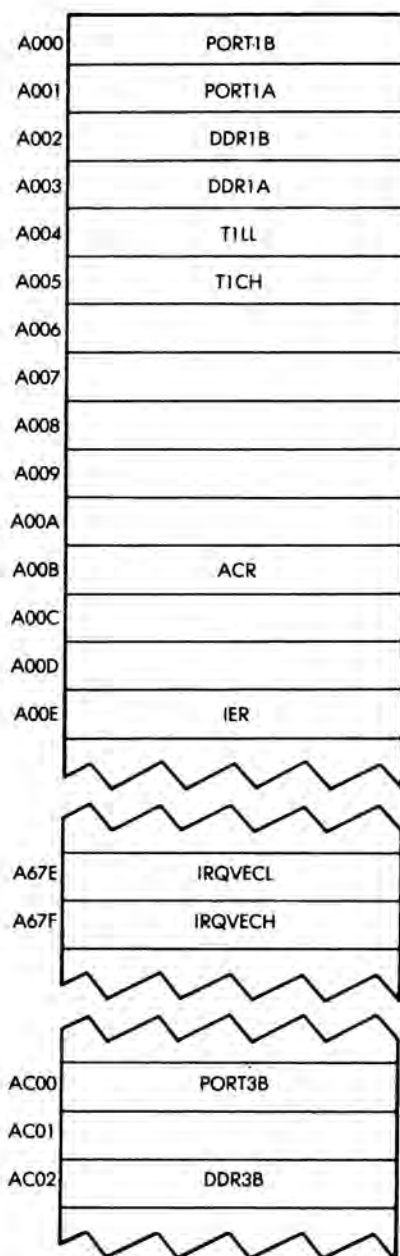


Figura 9.7: Mappa della memoria alta

## IL PROGRAMMA

### Strutture dei dati

Vengono impiegate due tabelle a nove ingressi per memorizzare, rispettivamente, la sequenza del computer e la sequenza del giocatore. Esse sono memorizzate a partire dagli indirizzi DIG0 e ENTRY0. (Vedere Fig. 9.6).

### Le variabili

Come al solito, viene impiegata la pagina 0 per fornire i registri di lavoro addizionali, cioè per memorizzare le variabili di lavoro. L'impiego della pagina 0 è indicato come "mappa di memoria" in Fig. 9.6. Le prime nove locazioni vengono impiegate per le variabili del programma. La funzione di ciascuna variabile è indicata nell'illustrazione e sarà descritta in dettaglio in seguito nel corso dell'esame del programma. Le locazioni da "09" a "0E" sono riservate per la tabella di numeri casuali impiegata per generare i numeri casuali. Le locazioni da "0F" a "17" sono impiegate dalla tabella DIG0 per memorizzare la sequenza generata dal computer. Infine, le locazioni da "18" in poi sono impiegate per contenere la sequenza di digit caricata dall'utente.

La Fig. 9.7 mostra le locazioni di memoria impiegate per l'indirizzamento d'ingresso/uscita e per la vettorizzazione degli interrupt. Le locazioni da "A000" ad "A005" sono impiegate per indirizzare le Porte A e B del VIA # 1 come pure del timer T1. La Fig. 9.8 mostra la mappa di memoria di un 6522 VIA.

La locazione "A00B" viene impiegata per accedere al registro di controllo ausiliario, mentre la locazione "A00E" accede al registro di abilitazione interrupt. Per una descrizione dettagliata di questi registri si rimanda il lettore ad *Applicazioni del 6502* pure edito dal Gruppo Editoriale Jackson.

Le locazioni di memoria "A67E" ed "A67F" sono impiegate per predisporre il vettore d'interrupt. L'indirizzo di partenza della routine di manipolazione interrupt sarà memorizzato in questa locazione di memoria. Nel nostro programma questo indirizzo sarà "03EA". Questa è la routine che genera il lampeggio dei LED; essa verrà descritta in seguito. Infine la Porta 3 viene indirizzata alle locazioni di memoria "AC00" ed "AC02".

### Implementazione del programma

La Fig. 9.9 riporta un diagramma di flusso dettagliato del programma

00	ORB (DA PB0 A PB7)	Dati di I/O, Porta A
01	ORA (DA PA0 A PA7)	Impiegato per il controllo - Influenza handshake
02	DDRB	Registri di direzione dati
03	DDRA	
04	T1L-L/T1C-L	Contatore basso
05	T1C-H	Contatore alto
06	T1L-L	Latch basso
07	T1L-H	Latch alto
08	T2L-L/T2C-L	Latch basso
09	T2C-H	Latch alto
0A	SR	Registro di scorrimento
0B	ACR	Ausiliari
0C	PCR (CA1,CA2,CB2,CB1)	Periferiche
0D	IFR	Flag
0E	IER	Abilitazione
0F	ORA	Uscita registro A (non influenza handshake)

Figura 9.8: Mappa di memoria del 6522 VIA

Mindbender. Esaminiamo ora il programma stesso. (Vedere Fig. 9.13).

Il blocco di inizializzazione risiede agli indirizzi di memoria 0200 - 0239 esadecimali assieme alle locazioni di interrupt e di I/O. Innanzi tutto vengono condizionati gli interrupt. Prima di modificare il vettore di interrupt che risiede agli indirizzi "A67E" ed "A67F" (vedere Fig. 9.7) deve essere autorizzato l'accesso a questa area di memoria protetta. Questo viene eseguito dalla subroutine ACCESS, che è una parte del SYM monitor:

#### JSR ACCESS

Successivamente, si può caricare il nuovo vettore d'interrupt alla locazione specifica. Il valore "03EA" viene caricato all'indirizzo IRQVEC:

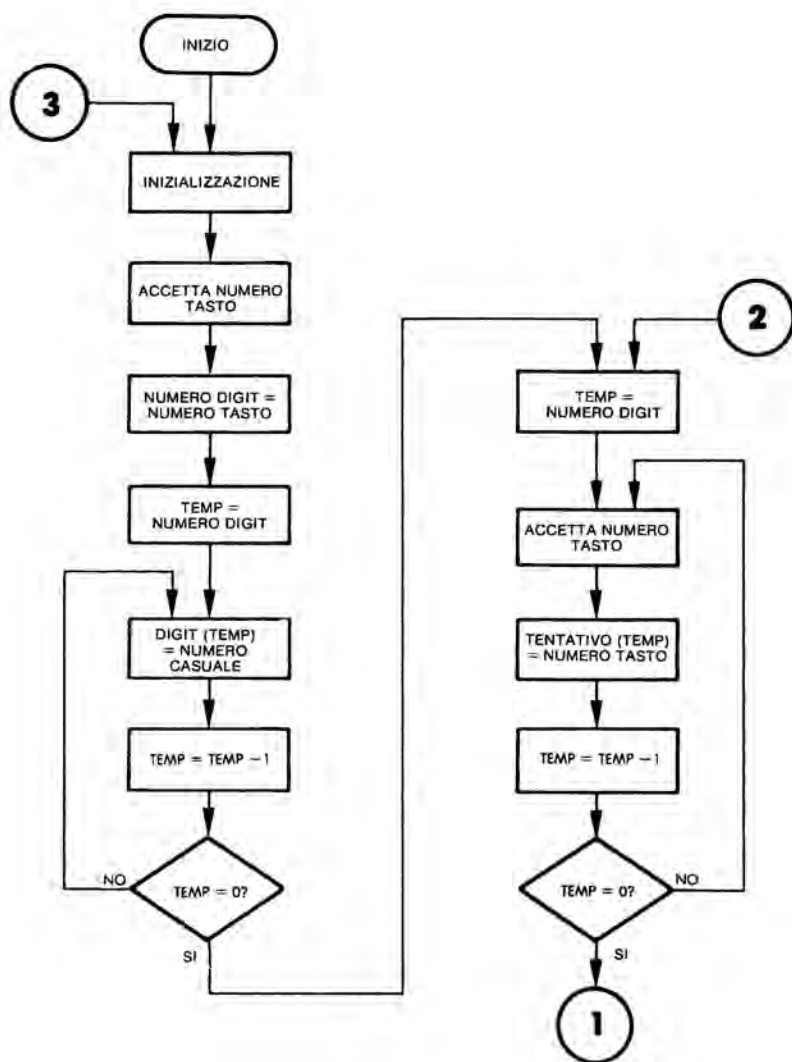


Figura 9.9: Diagramma di flusso dettagliato del Mindbender

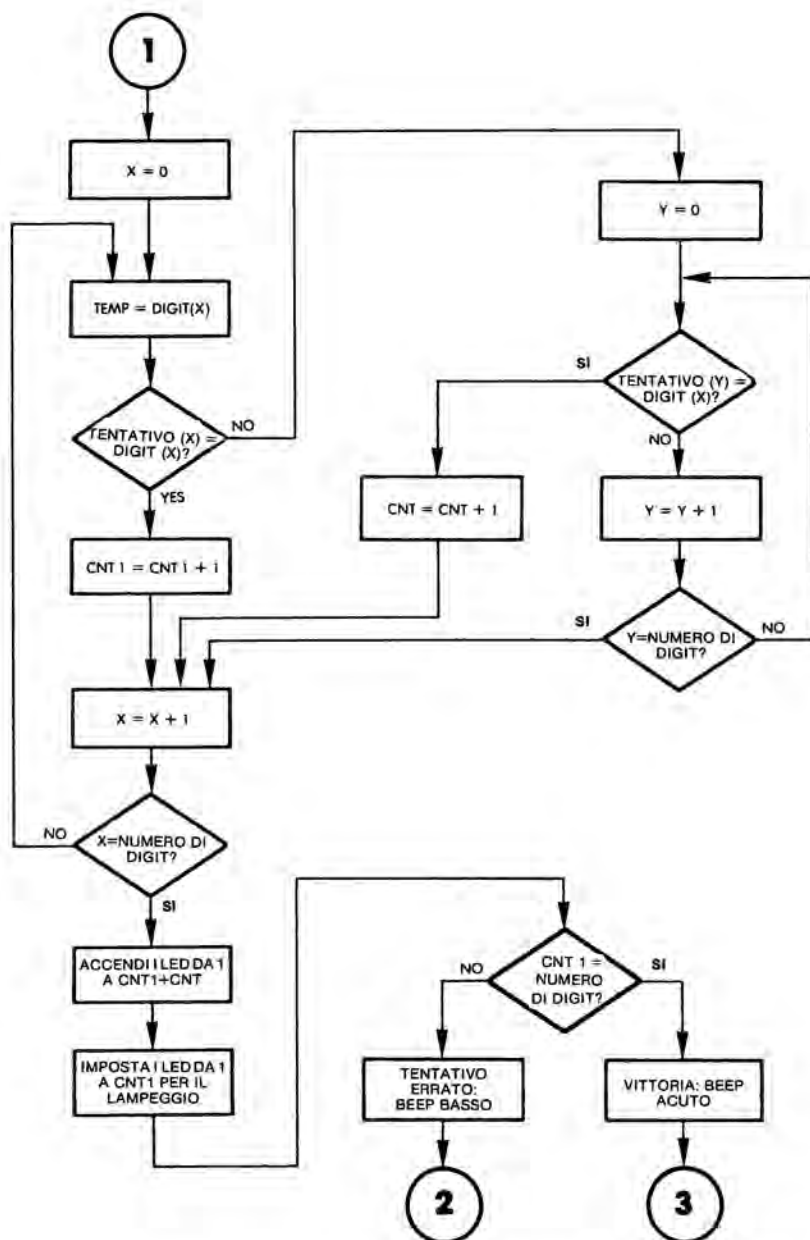


Figura 9.9: Diagramma di flusso dettagliato del Mindbender (continua)

LDA #\$EA	Vettore d'interrupt basso
STA IRQVECL	
LDA #\$03	Vettore d'interrupt alto
STA IRQVECH	

Ora occorre condizionare i registri interni del 6522 VIA # 1 in modo da predisporre gli interrupt. Il registro di abilitazione interrupt (IER) imporrà l'abilitazione o disabilitazione degli interrupt. Ogni posizione di bit in IER accoppia quella corrispondente nel registro del flag di interrupt (IFR). Ogni volta che una posizione di bit è "0", l'interrupt corrispondente è disabilitato. Il bit 7 di IER ha un ruolo particolare. (Vedere Fig. 9.10). Quando il bit 7 di IER è "0", ogni "1" nelle posizioni di bit rimanenti azzererà il corrispondente flag di abilitazione. Quando il bit 7 di IER è "1", ogni "1" scritto in IER si comporterà normalmente ed imporrà un'abilitazione. Quindi, tutti gli interrupt sono disabilitati ponendo il bit 7 a "0" e tutti i bit rimanenti di IER ad 1:

```
LDA #$7F
STA IER
```

Successivamente viene abilitato il bit 6, che corrisponde all'interrupt del timer 1. Per eseguire questo, il bit 7 di IER viene posto ad "1", come pure il bit 6:

```
LDA #$C0
STA IER
```

Successivamente il timer 1 sarà posto nel modo "free-running" (a corsa libera). Ricordate che, con il 6522, il timer può essere impiegato sia nel modo "one-shot" (un colpo) che nel modo "free-running". I bit 6 e 7 del registro di controllo ausiliario vengono impiegati per selezionare i modi operativi del timer 1. (Vedere Fig. 9.11). In questo caso, il bit 7 è posto a 0 ed il bit 6 a 1:

```
LDA #$40
STA ACR
```

Prima dell'impiego del timer nel modo d'uscita, il suo registro contatore deve essere caricato con un valore a 16 bit. Questo valore specifica la

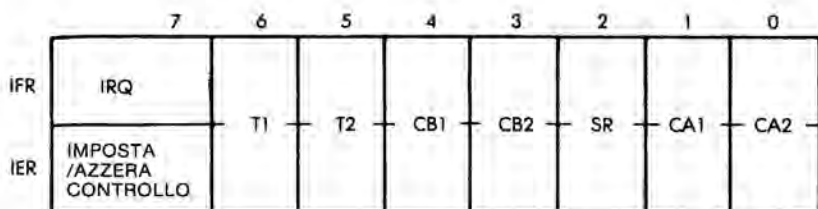


Figura 9.10: Registri di interrupt

durata dell'impulso ad onda quadra da generare. In questo caso viene impiegato il massimo valore "FFFF":

```
LDA #$FF
STA T1LL
STA T1CH
```

La Fig. 9.12 mostra la forma d'onda reale proveniente dal timer 1. Per calcolare la durata esatta dell'impulso, notate che la durata dell'impulso si alternerà tra  $n + 1,5$  cicli ed  $n + 2$  cicli, essendo  $n$  il valore caricato inizialmente nel registro contatore.

Successivamente vengono abilitati gli interrupt:

```
CLI
```

e le tre porte impiegate da questo programma sono configurate nella

ACR7 ABILITA- ZIONE D'USCITA	ACR6 ABILITA- ZIONE D'INGRESSO	MODO
0	0 (ONE-SHOT)	GENERA TIME OUT INT QUANDO T1 VIENE CARICATO IN PB7 DISABILITATA
0	1 (FREE RUN)	GENERA INT CONTINUI SU PB7 DISABILITATA
1	0 (ONE-SHOT)	GENERA INT ED IMPULSO D'USCITA SU PB7 OGNI VOLTA CHE T1 VIENE CARICATO = ONE-SHOT ED IMPULSO DI LARGHEZZA PROGRAMMABILE
1	1 (FREE RUN)	GENERA INT CONTINUI ED USCITA AD ONDA QUADRA SU PB7

Figura 9.11: Il registro di controllo ausiliario del 6522 seleziona i modi di funzionamento del timer 1

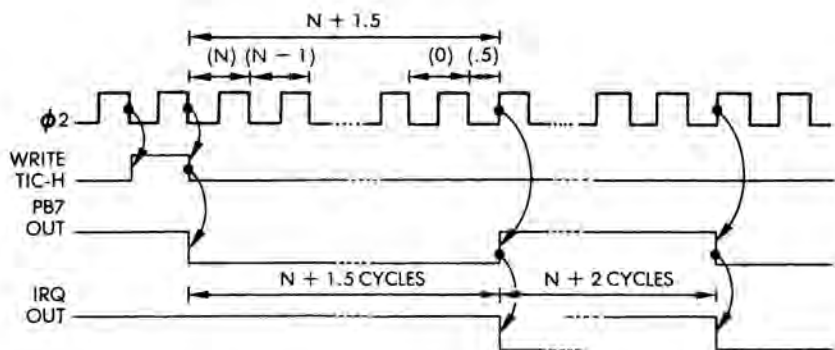


Figura 9.12: Il timer 1 funziona nel modo Free Running

direzione appropriata:

STA DDR1A	Uscita
STA DDR1B	Uscita
STA DDR3B	Uscita

Tutti i LED vengono quindi spenti:

```
KEY1      LDA #0
           STA PORT1A
           STA PORT1B
```

e le maschere di lampeggio sono inizialmente poste a tutti 0:

```
STA MASKA
STA MASKB
```

Il LED 10 viene ora acceso per segnalare al giocatore che deve specificare il numero di digit da indovinare:

LDA #%00000010	Scegli il LED 10
STA PORT1B	Commutalo in on

Il tasto premuto viene letto impiegando la solita routine GETKEY:

JSR GETKEY	Accetta il # di digit
------------	-----------------------



A questo punto viene implementato un filtro software. Il valore del tasto letto dalla tastiera viene convalidato se cade nella gamma da "1" a "9". Se esso è maggiore di 9, oppure inferiore ad 1, l'ingresso viene ignorato:

```
CPM #10
BPL KEY1
CMP #0
BEQ KEY1
```

Una volta convalidata, la lunghezza specificata dalla sequenza viene memorizzata alla locazione di memoria DIGITS:

STA DIGITS

A questo punto deve essere generata una sequenza di numeri casuali.

#### *Generazione di una sequenza di numeri casuali*

Il numero casuale iniziale viene ottenuto dal contatore ed impiegato per avviare il generatore di numeri casuali. La teoria alla base di questa tecnica è stata precedentemente descritta.

Le locazioni RND + 1, RND + 4 ed RND + 5 vengono caricate con lo stesso numero:

```
LDA TILL
STA RND + 1
STA RND + 4
STA RND + 5
```

Quindi si ottiene un numero casuale impiegando la subroutine RANDOM:

	LDY DIGITS	Accetta il # di digit da indovinare
	DEY	Conteggio fino a 0
RAND	JSR RANDOM	Riempili con i valori

Il numero casuale risultante viene posto ad un valore BCD che garantisce che l'ultimo digit sarà compreso tra 0 e 9:

```
SED
ADC #00
CLD
```

Aggiustamento decimale

Esso viene quindi troncato ai quattro bit inferiori:

```
AND #$00001111
```

Una volta che è stato ottenuto il digit casuale, viene salvato nella locazione successiva della tabella dei digit, impiegando il registro indice Y come puntatore corrente:

```
STA DIG0,Y
```

Il contatore Y viene quindi decrementato ed il ciclo viene eseguito fino a che sono stati generati tutti i digit richiesti:

```
DEY  
BPL RAND
```

#### *Raccolta dei tentativi del giocatore*

Il registro indice X servirà come puntatore corrente per la tabella ENTRY impiegata per raccogliere i tentativi del giocatore. Esso viene inizializzato al valore "0" e memorizzato alla locazione di memoria XTEMP:

```
EXTRA      LDA #0           Azzera il puntatore  
           STA XTEMP
```

Vengono quindi accesi i LED 10 ed 11 per segnalare al giocatore che può caricare la propria sequenza:

```
LDA #$00000110  
STA PORT1B
```

Il tasto premuto dal giocatore viene letto mediante la solita routine GETKEY:

```
KEY2      JSR GETKEY
```

Se il tasto premuto è maggiore di 9, esso viene interpretato come una richiesta di ri-inizializzazione del gioco:

```
CMP #10  
BPL KEY1
```

Altrimenti viene recuperato dalla locazione di memoria XTEMP il valore del registro indice e viene impiegato per eseguire la memorizzazione indicizzata dell'accumulatore nella locazione appropriata della tabella ENTRY:

LDX XTEMP	
STA ENTRY0,X	Memorizza il tentativo in tabella

Il puntatore corrente viene quindi incrementato e ri-memorizzato in memoria:

INX
STX XTEMP

Quindi, il valore del puntatore corrente viene confrontato con il numero massimo di digit da prelevare dalla tastiera e, finchè non viene raggiunto questo numero, il ciclo continua ritornando indietro alla locazione KEY2:

CPX DIGITS	Prelevati tutti i numeri?
BNE KEY2	Se no, accettane un altro

Una volta che il giocatore ha introdotto la propria sequenza, i digit devono essere confrontati con la sequenza generata dal computer. In previsione della visualizzazione di una possibile vincita i LED della scheda vengono spenti e le maschere azzerate:

LDX #0
STX PORT1A
STX PORT1B
STX MASKA
STX MASKB

Due locazioni della memoria saranno impiegate per contenere il numero di digit corretti ed il numero di digit corretti nella locazione corretta. Queste locazioni vengono inizialmente azzerate:

STX CNT	Numero di uguaglianze
STX CNT1	Numero di digit corretti

Ciascun ingresso della tabella DIG0 verrà ora confrontato con tutti gli ingressi della tabella ENTRY0. Ogni digit viene caricato dalla tabella

DIGIT e confrontato immediatamente con i contenuti corrispondenti in ENTRY:

```
DIGLP      LDA DIG0,X  
            CMP ENTRY0,X
```

Se non era il digit corretto al posto corretto, non c'è uguaglianza. Quindi controlleremo se il digit appare in qualche altra posizione all'interno della tabella ENTRY:

```
BNE ENTRYCMP
```

Altrimenti, viene registrata un'uguaglianza incrementando la locazione CNT1 e viene esaminato il digit successivo:

```
INC CNT1  
BNE NEXTDIG
```

Esaminiamo ora cosa succede quando non si verifica alcuna uguaglianza. Il digit (del numero da indovinare) che è stato appena letto ed è contenuto nell'accumulatore dovrebbe essere confrontato con ogni digit della tabella ENTRY. Il registro indice Y viene impiegato come puntatore corrente ed i contenuti dell'accumulatore sono confrontati a ciascuno dei digit di ENTRY:

```
ENTRYCMP   LDY #0  
ENTRYLP    CMP ENTRY0,Y  
            BNE NEXTENT
```

Se si trova un'uguaglianza, la locazione di memoria CNT viene incrementata e viene esaminato il digit successivo:

```
INC CNT  
BNE NEXTDIG
```

Altrimenti viene incrementato il registro indice Y. Se è stata raggiunta la fine della sequenza, si esce a NEXDIG. Altrimenti si verifica una ramificazione indietro all'inizio del ciclo alla locazione ENTRYLP:

```
NEXTENT    INY                Incrementa il puntatore del #  
                                di tentativo
```

CPY DIGITS  
BNE ENTRYLP

Tutti controllati?  
No: prova il successivo

Deve essere quindi esaminato il digit successivo nella tabella DIG. Il puntatore corrente di DIG è contenuto nel registro indice X. Esso viene incrementato e confrontato con il suo valore massimo:

NEXTDIG	INX	Incrementa il puntatore del # di digit
	CPX DIGITS	Controllati tutti i digit

Se il limite non è stato raggiunto, si ha una ramificazione indietro all'inizio del ciclo esterno alla locazione DIGLP:

BNE DIGLP

A questo punto siamo pronti per accendere i LED che visualizzano il risultato al giocatore.

#### *Visualizzazione del risultato al giocatore*

Il numero totale di LED che devono essere accesi viene ottenuto sommando i contenuti di CNT a CNT1:

CLC	Accetta il via per la somma
LDA CNT	
ADC CNT1	

Il totale è contenuto nell'accumulatore e trasferito nel registro indice Y dove sarà utilizzato dalla routine LITE:

TAY  
JSR LITE

Il funzionamento della routine LITE verrà descritto in seguito. Il suo effetto è di riempire l'accumulatore con il numero appropriato di uni in modo da accendere i LED corretti.

La struttura creata dalla subroutine LITE viene quindi memorizzata nella maschera:

STA PORT1A

Per il caso particolare in cui il risultato è 9, il bit carry sarà posto ad 1. Questo caso viene controllato esplicitamente:

BCC CC	Se carry è 0, non accendere PB0.
--------	-------------------------------------

e, se il carry è stato posto ad 1, la Porta B sarà posta in modo tale che il LED # 9 sia acceso:

LDA #1	Accendi PB0
STA PORT1B	

Ricordate che, una volta che le maschere A e B sono state predisposte, esse verranno usate automaticamente dalla routine di manipolazione interrupt che genererà il lampeggio dei LED appropriati:

CC	LDY CNT1
	JSR LITE
	STA MASKA
	BCC TEST
	LDA #01
	STA MASKB

Il programma a questo punto deve eseguire il test per vedere se si tratta di una situazione di vittoria o di sconfitta.

*Controllo situazione di vincita o di perdita*

Il numero di digit corretti nella posizione corretta è contenuto in CNT1. Confronteremo semplicemente questo valore con la lunghezza della sequenza da indovinare:

TEST	LDX CNT1
	CPX DIGITS

Se questi numeri sono uguali, il giocatore ha vinto:

BEQ WIN

Altrimenti viene eseguito un suono basso. La costante di durata del suono è posta a "72" ed il suo valore di frequenza a "BE":

```
BAD      LDA #$72
          STA DUR
          LDA #$BE
```

Per generare il suono verrà quindi impiegata la subroutine TONE, come al solito:

```
JSR TONE
```

Quindi si verifica un ritorno all'inizio del programma:

```
BEQ ENTER
```

Se si è verificata una vittoria, verrà generato un suono di elevata intensità. La sua costante di durata è posta ad "FF" e la sua intensità è controllata ponendo la sua costante di frequenza a "54":

```
WIN      LDA #$FF
          STA DUR
          LDA #$54
```

Come al solito, per generare il suono viene impiegata la subroutine TONE:

```
JSR TONE
```

Il gioco viene quindi ri-inizializzato:

```
JMP KEY1
```

### **Le subroutine**

Questo programma impiega quattro subroutine. Queste sono: LITE, RANDOM, TONE ed INTERRUPT HANDLER. Le subroutine RANDOM e TONE sono già state descritte ai capitoli precedenti e non verranno considerate in questa sede.

#### *Subroutine LITE*

All'ingresso in questa subroutine, il registro indice Y contiene il numero dei LED che dovrebbero lampeggiare. Per farli lampeggiare è

necessario caricare opportunamente le configurazioni a maschera chiamate MASKA e MASKB. In queste due locazioni deve essere caricato il numero appropriato di uni. Preliminarmente viene eseguito un test per vedere se in Y è contenuto il valore "0". Se si trova questo valore, l'accumulatore viene azzerato, come pure il bit carry (che verrà impiegato come indicatore del fatto che Y contiene il valore 9):

LITE	BNE STRTSH	Test di Y per zero
	LDA #0	
	CLC	
	RTS	

Altrimenti l'accumulatore viene inizialmente azzerato ed il numero appropriato di uni viene fatto scorrere a sinistra nell'accumulatore stesso attraverso il bit carry. Gli uni vengono caricati uno alla volta ponendo ad uno il bit carry e quindi eseguendo uno scorrimento a sinistra in A. Ogni volta il registro indice Y viene decrementato ed il ciclo viene ri-eseguito finchè Y non è "0":

	LDA #0	
SHIFT	SEC	
	ROL A	Scorrimento in posizione
	DEY	
	BNE SHIFT	Ciclo
	RTS	

Notate che viene impiegata una rotazione a sinistra anzichè uno scorrimento. Se Y non contiene il valore "9", l'accumulatore A dovrebbe essere riempito con uni ed il carry dovrebbe contenere il valore "1" all'uscita dalla subroutine.

### *Il manipolatore di interrupt*

Questa subroutine complementa i LED ogni volta che viene ricevuto un interrupt, cioè ogni volta che si ha il segnale dal timer 1. Essa è localizzata in memoria agli indirizzi "03EA" e successivi. Poichè l'accumulatore è impiegato come registro di lavoro dalla subroutine, all'atto dell'ingresso esso deve essere preservato e spinto nello stack:

PHA

I contenuti delle Porte 1A ed 1B saranno letti e quindi complementati.



Ricordate che non esiste un'istruzione di complemento nel 6502, e quindi al suo posto verrà impiegata un'operazione di OR inclusivo. MASKA e MASKB specificano i bit che devono essere complementati:

```
LDA PORT1A
EOR MASKA
STA PORT1A
LDA PORT1B
EOR MASKB
STA PORT1B
```

Inoltre ricordate che il bit di interrupt del 6522 deve essere esplicitamente azzerato dopo ogni interrupt. Questo viene eseguito leggendo il latch:

```
LDA TILL
```

Infine viene ripristinato l'accumulatore e si ha il ritorno al programma principale:

```
PLA
RTI
```

## SOMMARIO

In questo programma abbiamo impiegato due nuove risorse hardware del chip di I/O 6522: il controllo di interrupt ed il temporizzatore ad intervallo programmabile. Gli interrupt sono stati impiegati per implementare elaborazioni contemporanee di lampeggio dei LED mentre il programma procede, verificando una situazione di vittoria o sconfitta.

**Esercizio 9.1:** *Siete in grado di implementare lo stesso programma senza l'impiego di interrupt?*

```

PROGRAMMA MINDBENDER ESECUZIONE DEL GIOCO MINDBEN-
DER: L'UTENTE SPECIFICA LA LUNGHEZZA DEL NUMERO DA INDO-
VINARE, QUINDI INDOVINA I DIGIT ED IL COMPUTER RISPONDE IN-
DICANDO QUANTI DIGIT ERANO CORRETTI, QUELLI NON CORRETTI
E LE RELATIVE POSIZIONI, FINCHÉ IL GIOCATORE PUO' INDOVINA-
RE. SULLA SCHEDA, I LED LAMPEGGIANTI INDICANO IL DIGIT COR-
RETTI, I LED NON LAMPEGGIANTI INDICANO IL DIGIT CORRETTO
MA CON POSIZIONE ERRATA. LA RIGA IN BASSO DI LED VIENE IMPIE-
GATA PER VISUALIZZARE IL MODO DEL PROGRAMMA: SE È ACCESO
IL LED DI ESTREMA SINISTRA, IL PROGRAMMA ATTENDE CHE
L'UTENTE CARICHI LA LUNGHEZZA DEL NUMERO DA INDOVINARE.
SE SONO ACCESI I DUE LED DI ESTREMA SINISTRA, IL PROGRAMMA
ATTENDE UN TENTATIVO. IL PROGRAMMA RESPINGE I VALORI NON
CORRETTI PER LA LUNGHEZZA DEL NUMERO, CHE POSSONO ESSE-
RE SOLO DA 1 A 9. UN VALORE DI TENTATIVO DIVERSO DA 0-9, RI-INI-
ZIALIZZA IL GIOCO. UN SUONO BASSO INDICA UN TENTATIVO ER-
RATO, UN SUONO ALTO UNA VINCITA DOPO UNA VINCITA, IL PRO-
GRAMMA SI RI-INIZIALIZZA. PER IL LAMPEGGIO DEI LED VIENE IM-
PIEGATA UNA ROUTINE DI INTERRUPT.

```

```

      = $ 200
GETKEY = $ 100
ACCESS = $ 8B86

```

```

DIGITS = $ 00
DUR     = $ 01
XTEMP  = $ 02
XTEMP  = $ 03
CNT     = $ 04

```

```

MASKA = $ 05

```

```

MASKB = $ 06

```

```

FREQ   = $ 07

```

```

CNT1   = $ 08

```

```

RND     = $ 09

```

```

DIGO    = $ 0F

```

```

ENTRY0  = $ 18

```

```

IRQVECL = $ A67E

```

```

IRQVECH = $ A67F

```

```

IER      = $ A00E

```

```

ACR      = $ A00B

```

```

T1LL     = $ A004

```

```

T1CH     = $ A005

```

```

PORT1A   = $ A001

```

```

ROUTINE DELLA MEMORIA DI SISTEMA
NON PROTETTA
NUMERO DI DIGIT DA INDOVINARE
CONSTANTE DI DURATA DEL SUONO
MEMORIA TEMPORANEA PER IL REG. X.
MEMORIA TEMPORANEA PER IL REG. Y.
CONSERVA TRACCIA DEL # DI
UGUAGLIANZE
CONTIENE LA STRUTTURA PER L'EOR
DEI LED
REGISTRO DI STATO A PER IL
LAMPEGGIO
MASCHERA DI LAMPEGGIO DEI LED
DELLA PORTA B
MEMORIZZAZIONE TEMPORANEA PER LA
FREQUENZA DEL SUONO
# DI DIGIT CORRETTI IN POSIZIONE
CORRETTA
PRIMA LOCAZIONE DEI NUMERI CASUALI
PRIMA DELLE 9 LOCAZIONI DEI DIGIT
PRIMA DELLE 9 LOCAZIONI
DEI TENTATIVI
BYTE DI BASSO ORDINE DEL
VETTORE DI INTERRUPT
....E DI ORDINE ELEVATO
REGISTRI DEL 6522 VIA #1:
REGISTRO ABILITAZIONE INTERRUPT
REGISTRO DI CONTROLLO AUSILIARIO
LATCH BASSO DEL TIMER 1
CONTATORE ALTO DEL TIMER 1
REGISTRO DI I/O DELLA PORTA VIA 1

```

Figura 9.13: Programma del Mindbender

```

DDR1A  = $ A003      ; REGISTRO DI DIREZIONE DATI DELLA
                        ; PORTA A DEL VIA 1
PORT1B  = $ A000      ; REGISTRO DI I/O DELLA PORTA B VIA 1
DDR1B   = $ A002      ; REGISTRO DI DIREZIONE DATI DELLA
                        ; PORTA B VIA 1
PORT3B  = $ AC00      ; REGISTRO DI I/O DELLA PORTA
                        ; B VIA 3
DDR3B   = $ AC02      ; REGISTRO DI DIREZIONE DATI DELLA
                        ; PORTA B VIA 3

```

ROUTINE PER IMPOSTARE LE VARIABILI ED IL TIMER DI  
INTERRUPT PER IL LAMPEGGIO DEI LED

```

0200: 20 86 8B      JSR ACCESS      ; MEMORIA DI SISTEMA NON PROTETTA
0203: A9 EA          LDA #$EA        ; CARICA IL VETTORE BASSO DI INTERRUPT
0205: 8D 7E A6      STA IRQVECL     ; .....E MEMORIZZALO ALLA LOCAZIONE DEL
                                ; VETTORE
0208: A9 03          LDA #$03        ; CARICA IL VETTORE DI INTERRUPT.....
020A: 8D 7F A6      STA IRQVECH     ; ..... E MEMORIZZALO.
020D: A9 7F          LDA #$7F        ; AZZERA IL REGISTRO DI ABILITAZIONE
                                ; INTERRUPT
020F: 8D 0E A0      STA IER          ;
0212: A9 C0          LDA #$C0        ; ABILITA L'INTERRUPT DEL TIMER 1
0214: 8D 0E A0      STA IER          ;
0217: A9 40          LDA #$40        ; ABILITA IL TIMER 1 NEL MODO A
                                ; CORSA LIBERA
0219: 8D 0B A0      STA ACR          ;
021C: A9 FF          LDA #$FF        ;
021E: 8D 04 A0      STA T1LL        ; IMPOSTA LATCH BASSO SUL TIMER 1
0221: 8D 05 A0      STA T1CH        ; IMPOSTA LATCH ALTO ED INIZIA
                                ; CONTEGGIO
0224: 58             CLI             ; ABILITA GLI INTERRUPT
0225: 8D 03 A0      STA DDR1A        ; CONFIGURA COME USCITA LA PORTA A
                                ; DEL VIA 1
0228: 8D 02 A0      STA DDR1B        ; CONFIGURA COME USCITA LA PORTA B
                                ; DEL VIA 1
022B: 8D 02 AC      STA DDR3B        ; CONFIGURA COME USCITA LA PORTA B
                                ; DEL VIA 3
022E: A9 00          KEY1 LDA # 0     ; SPEGNI I LED
0230: 8D 01 A0      STA PORT1A      ;
0233: 8D 00 A0      STA PORT1B      ;
0236: 85 05          STA MASKA       ;
0238: 85 06          STA MASKB       ; AZZERA LE MASCHERE DI LAMPEGGIO

```

ROUTINE PER ACCETTARE IL NUMERO DI DIGIT DA INDOVINARE  
QUINDI RIEMPIRE I DIGIT CON NUMERI CASUALI DA 0 A 9

```

023A: A9 02          LDA # %00000010 ; ACCENDI IL LED PER SEGNALARE
                                ; ALL'UTENTE DI
023C: 8D 00 A0      STA PORT1B      ; CARICARE IL NUMERO DI DIGIT
                                ; RICHIESTI.
023F: 20 00 01      JSR GETKEY      ; ACCETTA IL # DI DIGIT RICHIESTI
0242: C9 0A          CMP # 10        ; SE IL NUMERO TASTO : 9, RESTART
                                ; DEL GIOCO
0244: 10 E8          BPL KEY1        ;
0246: C9 00          CMP # 0         ; CONTROLLO PER 0 DIGIT DA INDOVINARE
                                ; .....0 DIGIT NON CONSENTITI
0248: F0 E4          BEQ KEY1        ; MEMORIZZA IL NUMERO VALIDO DI DIGIT
024A: 85 00          STA DIGITS      ;
024C: AD 04 A0      LDA T1LL        ; ACCETTA # CASUALE
024F: 85 0A          STA RND + 1     ; USALO PER AVVIARE IL GENERATORE
0251: 85 0D          STA RND + 4     ; DI NUMERI CASUALI.
0253: 85 0E          STA RND + 5

```

Figura 9.13: Programma del Mindbender (continua)

```

0255: A4 00      LDY DIGITS      ; ACCETTA # DI DIGIT DA INDOVINARE,
0257: 88          DEY              ; ...CONTEGGIO FINO A 0, RIEMPENDOLI
                                ; CON VALORI.
0258: 20 FF 02 RAND JSR RANDOM    ; ACCETTA IL VALORE CASUALE DEL DIGIT
025B: F8          SED
025C: 69 00      ADC # 00         ; AGGIUSTAMENTO DECIMALE
025E: D8          CLD
025F: 29 0F      AND # %00001111; MANTIENI DIGIT <10
0261: 99 0F 00   STA DIGO, Y     ; SALVALO NELLA TABELLA DEI DIGIT.
0264: 88          DEY
0265: 10 F1      BPL RAND        ; RIEMPI IL DIGIT SUCCESSIVO

;
; ROUTINE PER RIEMPIRE LA TABELLA DEI TENTATIVI CON I
; TENTATIVI DELL'UTENTE
;
0267: A9 00      ENTER LDA # 0     ; AZZERA IL PUNTATORE D'INGRESSO
                                ; ALLA TABELLA
0269: 85 02      STA XTEMP
026B: A9 06      LDA # %0000110; INFORMA L'UTENTE QUALI TENTATIVI
026D: 0D 00 A0   ORA PORT1B      ; DOVREBBERO ESSERE CARICATI.....
0270: 8D 00 A0   STA PORT1B      ; ...SENZA VARIARE LA MATRICE
0273: 20 00 01 KEY2 JSR GETKEY    ; ACCETTA TENTATIVO
0276: C9 0A      CMP # 10        ; È MAGGIORE DI 9?
0278: 10 B4      BPL KEY1        ; SE SÌ, RESTART DEL GIOCO
027A: A6 02      LDX XTEMP       ; ACCETTA PUNTATORE PER
                                ; L'INDICIZZAZIONE
027C: 95 18      STA ENTRY0, X   ; MEMORIZZA TENTATIVO IN TABELLA
027E: E8          INX            ; INCREMENTA PUNTATORE
027F: 86 02      STX XTEMP
0281: E4 00      CPX DIGITS      ; PRELEVATO # CORRETTO DI TENTATIVI?
0283: D0 EE      BNE KEY2        ; SE NO, ACCETTANE UN ALTRO

; QUESTA ROUTINE CONFRONTA I TENTATIVI DELL'UTENTE CON I
; DIGIT DEL NUMERO DA INDOVINARE. PER OGNI DIGIT CORRETTO
; NELLA POSIZIONE CORRETTA, VIENE ACCESO UN LED LAMPEG-
; GIANTE E PER OGNI DIGIT CORRETTO NELLA POSIZIONE ERRATA,
; VIENE ACCESO UN LED NON LAMPEGGIANTE.
;
0285: A2 00      LDX # 0         ; AZZERA LE SEGUENTI MEMORIE:
0287: 8E 01 A0   STX PORT1A      ; LED
028A: 8E 00 A0   STX PORT1B
028D: 86 05      STX MASKA       ; MASCHERE DI LAMPEGGIO
028F: 86 06      STX MASKB
0291: 86 04      STX CNT
0293: 86 08      STX CNT1
0295: B5 0F      DIGLP LDA DIGO, X ; CARICA PER IL CONFRONTO IL PRIMO
                                ; DIGIT DEL #
0297: D5 18      CMP ENTRY0, X   ; TENTATIVO CORRETTO/POSIZIONE
                                ; CORRETTA?
0299: D0 04      BNE ENTRYCMP    ; NO: È DIGIT CORRETTO/POSIZIONE
                                ; ERRATA?
029B: E6 08      INC CNT1        ; UN ALTRO TENTATIVO CORRETTO
                                ; /POSIZIONE CORRETTA
029D: D0 10      BNE NEXTDIG     ; ESAMINA IL DIGIT SUCCESSIVO DEL
                                ; NUMERO
029F: A0 00      ENTRYCMP LDY # 0 ; RESET DEL PTR DEL # DI
                                ; TENTATIVO PER IL CONFRONTO
02A1: D9 19 00   ENTRYLPCMP ENTRY0, Y ; DIGIT CORRETTO/POSIZIONE ERRATA?
02A4: D0 04      BNE NEXTENT    ; NO, VEDI SE È IL DIGIT SUCCESSIVO
02A6: E6 04      INC CNT        ; UN ALTRO DIGIT CORRETTO/POSIZIONE
                                ; ERRATA

```

Figura 9.13: Programma del Mindbender (continua)

02A8: D0 05		BNE NEXTDIG	: ESAMINA IL DIGIT SUCCESSIVO DEL
02AA: C8	NEXTENTINY		: NUMERO
02AB: C4 00		CPY DIGITS	: INCREMENTA PTR DEL # DI
02AD: D0 F2		BNE ENTRYLP	: TENTATIVI
02AF: E8	NEXTDIGINX		: CONTROLLATI TUTTI I TENTATIVI?
02B0: E4 00		CPX DIGITS	: NO, VERIFICA IL TENTIVO SUCCESSIVO.
02B2: D0 E1		BNE DIGLP	: INCREMENTA PTR DEL # DI TENTATIVI
02B4: 18		CLC	: VALUTATI TUTTI I DIGIT?
02B5: A5 04		LDA CNT	: NO, CONTROLLA IL DIGIT SUCCESSIVO.
02B7: 65 08		ADC CNT1	: ACCETTA IL PRONTO PER.....
02B9: A8		TAY	: SOMMARE IL TOTALE DELLE
02BA: 20 F1 02		JSR LITE	: UGUAGLIANZE PER DETERMINARE
02BD: 8D 01 A0		STA PORT1A	: IL NUMERO DI LED DA ACCENDERE
02C0: 90 05		BCC CC	: TRASFERISCI A IN Y PER LA ROUTINE
02C2: A9 01		LDA # 1	: "LIGHT"
02C4: 8D 00 A0		STA PORT1B	: ACCETTA LA STRUTTURA DI LED DA
02C7: A4 08	CC	LDY CNT1	: ACCENDERE
02C9: 20 F1 02		JSR LITE	: ACCENDI I LED
02CC: 85 05		STA MASKA	: SE CARRY = 0, NON ACCENDERE PBO
02CE: 90 04		BCC TEST	: ACCENDI PBO
02D0: A9 01		LDA # 1	: CARICA # DI LED DA LAMPEGGIARE
02D2: 85 06		STA MASKB	: ACCETTA STRUTTURA
			: INIZIA IL LAMPEGGIO DEI LED
			: SE CARRY = 0, PBO NON LAMPEGGIA
			SUBROUTINE PER IL TEST DELLA VITTORIA MEDIANTE CONTROLLO
			SE IL NUMERO CORRETTO DI DIGIT IN POSIZIONE CORRETTA = NU-
			NUMERO DIGIT, IN CASO DI VITTORIA/ VIENE GENERATO UN SUONO
			ACUTO. MENTRE, SE UN DIGIT QUALSIASI È ERRATO, VIENE GENE-
			RATO UN SUONO BASSO.
02D4: A6 08	TEST	LDX CNT1 LOAD	: CARICA IL NUMERO DI DIGIT CORRETTI
02D6: E4 00		CPX DIGITS	: TUTTI TENTATIVI CORRETTI?
02D8: F0 08		BEQ WIN	: SE SI, IL GIOCATORE VINCE
02DA: A9 72	BAD	LDA # \$ 72	
02DC: 85 01		STA DUR	: IMPOSTA LA LUNGHEZZA DEL SUONO
02DE: A9 BE		LDA # \$ BE	: BASSO
02E0: 20 12 03		JSR TONE	: VALORE PER IL SUONO BASSO
02E3: F0 82		BEQ ENTER	: SEGNALE TENTATIVI ERRATI CON SUONO
02E5: A9 FF	WIN	LDA # \$ FF	: ACCETTA I TENTATIVI SUCCESSIVI
02E7: 85 01		STA DUR	: DURATA DEL SUONO ACUTO
02E9: A9 54		LDA # \$ 54	: VALORE PER SUONO ACUTO
02EB: 20 12 03		JSR TONE	: SEGNALE LA VITTORIA
02EE: 4C 2E 02		JMP KEY1	: RESTART DEL GIOCO
			ROUTINE PER RIEMPIRE L'ACCUMULATORE CON I BIT "1", INIZIAN-
			DO DALLA PARTE DI BASSO ORDINE, FINO AD ARRIVARE ALLA PO-
			SIZIONE DI BIT CORRISPONDENTE AL LED DA ACCENDERE O DA
			IMPOSTARE AL LAMPEGGIO.
02F1: D0 04	LITE	BNE STRTSH	: SE Y NON ZERO, FA SCORRERE GLI UNI
02F3: A9 00		LDA # 0	: CASO SPECIALE: RISULTATO NESSUN
02F5: 18		CLC	: UNO.
02F6: 60		RTS	
02F7: A9 00	STRTSH	LDA # 0	: AZZERA A COSICCHÈ VERRA' MOSTRATA
02F9: 38	SHIFT	SEC	: LA STRUTTURA
			: COSTRUISCI UN BIT 1

Figura 9.13: Programma del Mindbender (continua)

```

02FA: 2A          ROL A          ; FALLO SCORRERE NELLA POSIZIONE
                                ; CORRETTA
02FB: 88          DEY            ; MEDIANTE CICLO AL # DI
                                ; UGUAGLIANZA TENTATIVO/DIGIT,
                                ; PASSAGGIO IN Y
02FC: D0 FB      BNE SHIFT      ; RICICLA SE NON FATTO
02FE: 60          RTS

; GENERATORE DI NUMERI CASUALI USA I NUMERI A, B, C, D, E, F ME-
; MORIZZATI DA RND AD RND + 5: SOMMA B + D + E + F + 1 E CARICA IL
; RISULTATO IN A, QUINDI FA SCORRERE A IN B, B IN C, ECC. ALL'U-
; SCITA NELL'ACCUMULATORE SI TROVA IL NUOVO NUMERO CA-
; SUALE CHE È COMPRESO TRA 0 E 255

02FF: 38          RANDOMSEC      ; IL CARRY AGGIUNGE IL VALORE 1
0300: A5 0A      LDA RND + 1    ; SOMMA A, B, E E CARRY
0302: 65 0D      ADC RND + 4
0304: 65 0E      ADC RND + 5
0306: 85 09      STA RND
0308: A2 04      LDX # 4        ; FA SCORRERE I NUMERI
030A: B5 09      RPL LDA RND, X
030C: 95 0A      STA RND + 1, X
030E: CA          DEX
030F: 10 F9      BPL RPL
0311: 60          RTS

; ROUTINE PER LA GENERAZIONE DEL SUONO. ALLA CHIAMATA LA
; DURATA DEL SUONO (NUMERO DI CICLI DA GENERARE) DOVREBBE
; ESSERE UN "DUR" ED IL VALORE DELLA NOTA (FREQUENZA) NEL-
; L'ACCUMULATORE.

0312: 85 07      TONE STA FREQ
0314: A9 FF      LDA # $ FF
0316: 8D 00 AC   STA PORT3B
0319: A9 00      LDA # $ 00
031B: A6 01      LDX DUR
031D: A4 07      FL2 LDY FREQ
031F: 88          FL1 DEY
0320: 18          CLC
0321: 90 00      BCC + 2
0323: D0 FA      BNE FL1
0325: 49 FF      EOR # $ FF
0327: 8D 00 AC   STA PORT3B
032A: CA          DEX
032B: D0 F0      BNE FL2
032D: 60          RTS

; ROUTINE DI MANIPOLAZIONE INTERRUPT
; COMPLEMENTA I LED AD OGNI INTERRUPT

; = $3EA          ; POSIZIONA LA ROUTINE NELL'AREA ALTA
;                 ; DI MEMORIA
03EA: 48          PHA            ; SALVA L'ACCUMULATORE
03EB: AD 01 A0    LDA PORT1A    ; ACCETTA LA PORTA PER LA
                                ; COMPLEMENTAZIONE
03EE: 45 05      EOR MASKA      ; COMPLEMENTA I BIT NECESSARI
03F0: 8D 01 A0    STA PORT1A    ; MEMORIZZA I CONTENUTI
                                ; COMPLEMENTATI
03F3: AD 00 A0    LDA PORT1B    ; RIPETI PER LA PORTA 1B
03F6: 45 06      EOR MASKB

```

Figura 9.13: Programma del Mindbender (continua)

03F8: 8D 00 A0	STA	PORT1B	
03FB: AD 04 A0	LDA	T1LL	; AZZERA IL BIT DI INTERRUPT NEL VIA
03FE: 68	PLA		; RIMEMORIZZA L'ACCUMULATORE
03FF: 40	RTI		; FATTO, RITORNA AL PROGRAMMA

SYMBOL TABLE:

GETKEY	0100	ACCESS	8B85	DIGITS	0000
DUR	0001	XTEMP	0002	YTEMP	0003
CNT	0004	MASKA	0005	MASKB	0006
FREQ	0007	CNT1	0008	RND	0009
DIGO	000F	ENTRY0	0018	IRQVECL	A67E
IRQVECH	A67F	IER	A00E	ACR	A00B
T1LL	A004	T1CH	A005	PORT1A	A001
DDR1A	A003	PORT1B	A000	DDR1B	A002
PORT3B	AC00	DDR3B	AC02	KEY1	022E
RAND	0258	ENTER	0267	KEY2	0273
DIGLP	0295	ENTRYCMP	029F	ENTRYLP	02A1
NEXTENT	02AA	NEXTDIG	02AF	CC	02C7
TEST	02D4	BAD	02DA	WIN	02E5
LITE	02F1	STRTSH	02F7	SHIFT	02F9
RANDOM	02FF	RPL	030A	TONE	0312
FL2	031D	FL1	031F		
DONE					

Figura 9.13: Programma del Mindbender (continua)





## BLACKJACK

### LE REGOLE

Il Blackjack o "21" convenzionale viene eseguito nel modo seguente. Il giocatore, chiedendo una carta alla volta, con la somma dei valori delle carte in suo possesso, senza però superare i 21 punti, cerca di battere il punteggio totalizzato dal banco. Se in qualsiasi istante si ottiene il totale di 21, dopo aver giocato soltanto due carte, il giocatore ha automaticamente vinto; è chiamata un Blackjack (il nome del gioco). I valori delle carte vanno da 1 ad 11. Nella versione convenzionale del Blackjack le regole esigono che il banco "batta" (prenda una carta) se ha in mano 16 o meno punti, ma gli proibiscono la battuta se ha 17 o più punti.

La versione di Blackjack eseguita sulla Scheda Giochi è leggermente diversa dal gioco normale. Le carte usate qui hanno valori da 1 a 10 (anziché da 1 ad 11) ed il numero di punti non può superare 13 (al posto di 21). Il banco in questo caso è il computer.

All'inizio di ogni mano, viene assegnata una carta al banco ed una al giocatore. Un LED sempre acceso sulla Scheda Giochi rappresenta il valore della carta assegnata al banco (il computer). Un LED lampeggiante rappresenta la carta assegnata al giocatore. Se il giocatore vuole ricevere un'altra carta (battuta) deve premere il tasto "C". Il giocatore può battere diverse volte, comunque se il totale delle carte del giocatore dovesse superare 13 punti, il giocatore ha perso la mano ("ha sballato") e non può più giocare. È quindi il turno del banco. Analogamente, se il giocatore decide di passare, è il turno del banco. Il banco gioca nel modo seguente: se il totale in suo possesso è minore di 10 punti, il computer si attribuisce un'altra carta. Se non supera i 13 punti, controlla per vedere se è necessaria un'altra carta. Analogamente alla situazione del giocatore, il computer ha perso se il punteggio totale delle carte supera 13. Non è stato previsto nessun bonus o vittoria automatica ogni volta che il giocatore o il banco riesce ad ottenere esattamente 13 punti con due sole carte (un Blackjack). Questo viene lasciato come esercizio per il lettore.

Una volta che il banco termina il suo turno, supponendo che non sia sballato, i due punteggi vengono confrontati. Se il totale del banco è maggiore di quello del giocatore, il giocatore ha perso. In caso contrario il giocatore ha vinto. All'inizio di ogni serie al giocatore vengono assegnati 5 gettoni (5 punti). Ogni mano persa diminuisce questo totale di un gettone; ogni vincita lo aumenta di uno. Il gioco termina quando il giocatore ha perso tutto, oppure raggiunge un punteggio di 10 e vince. Dopo ogni gioco il punteggio risultante viene visualizzato sotto forma di numero tra 0 e 10 sul LED appropriato. Ogni volta che un giocatore vince una mano, vengono accesi i tre LED di sinistra della riga in basso. Se vince il banco, vengono accesi i tre LED di destra. (Vedere Fig. 10.1).

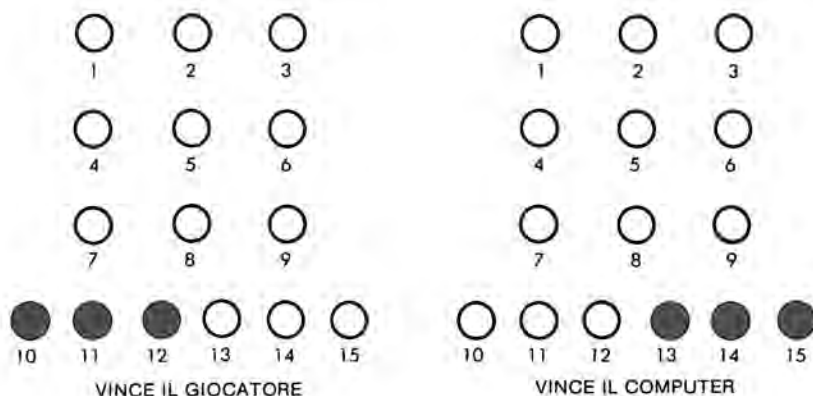


Fig. 10.1: Indicazione del vincitore

## UNA GIOCATA TIPICA

In una partita contro il banco, il giocatore preme il tasto A per ricevere un'altra carta (battuta) a meno che non superi il totale di 13 (bust) oppure a meno che il giocatore non decida che il suo totale è abbastanza vicino a 13 e sufficiente a battere il banco. Quando il giocatore prende la decisione di fermarsi, deve premere il tasto "C". Questo passa la mano al banco e tutti gli altri tasti verranno ignorati. I LED si accenderanno in successione sulla scheda quando il computer si assegna delle altre carte finché non supera il 10, raggiunge esattamente il 13, oppure sballa. Una volta che il computer ha arrestato il gioco, qualsiasi tasto può essere premuto; verrà visualizzato il punteggio del giocatore ed il vincitore verrà indicato accendendo i LED dal lato relativo. La visualizzazione apparirà per circa un secondo e quindi si passa ad una nuova mano.

Notate che, una volta che il valore in mano al computer ha raggiunto un totale maggiore o uguale a 10, esso non esegue nient'altro finché non viene premuto un tasto. Eseguiamo questa "giocata tipica".

La Fig. 10.2 mostra la visualizzazione iniziale. Un LED acceso viene rappresentato con un punto nero, mentre un LED lampeggiante viene rappresentato con mezzo punto. Nella mano iniziale, il computer ha assegnato a se stesso un 1 ed al giocatore un 4. Il giocatore preme il tasto "A" e riceve una carta addizionale. È un 9. La situazione è mostrata in Fig. 10.3. Si tratta di un Blackjack ed il giocatore ha vinto. La miglior speranza del banco a questo punto è di raggiungere anche lui 13.

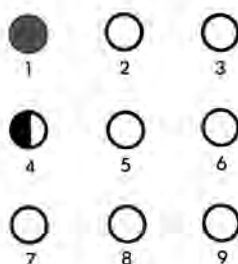


Figura 10.2: Prima mano

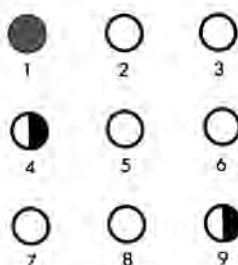


Figura 10.3: Il giocatore riceve una seconda carta: blackjack

Esaminiamo il suo responso. Per passare la mano dobbiamo premere il tasto "C". Un istante dopo si accende il LED # 3. Il totale in mano al computer è ora  $1 + 3 = 4$ . Esso si assegnerà un'altra carta. Un istante dopo si accende il LED # 7. Il totale del computer è ora  $4 + 7 = 11$ . Esso si arresta. Avendo un punteggio più basso del giocatore, ha perso. Verifichiamolo. Premiamo un tasto sulla tastiera (per esempio, "0"). Il

risultato appare sul display: si accendono i LED 10, 11 e 12 indicanti una vittoria del giocatore, inoltre si accende il LED # 6, indicando che il punteggio del giocatore è aumentato da 5 a 6 punti. Questa informazione è mostrata in Fig. 10.4.

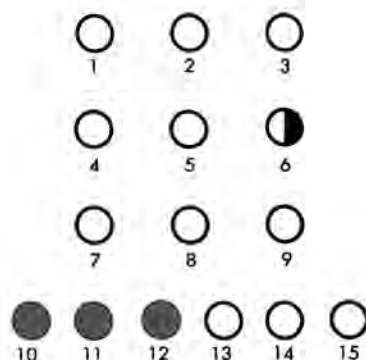


Figura 10.4: Fine del giro: il banco perde

Il visualizzatore quindi si spegne ed inizia una nuova mano. Quando viene eseguita l'estrazione di una carta, nessuno dei LED della riga in basso è acceso ed il punteggio rimane invariato. Viene eseguita una nuova mano. (Se il giocatore sballa, il banco vince immediatamente e viene visualizzata una vittoria del computer).

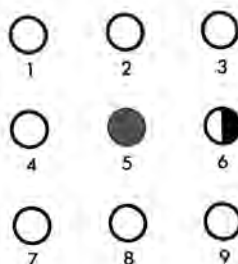


Figura 10.5: Seconda mano

Eseguiamo ora un'altra mano. All'inizio di questa mano il computer ha assegnato a se stesso un 5 ed al giocatore un 6. La situazione è mostrata in Fig. 10.5. Chiediamo un'altra carta. Premiamo il tasto "A" e

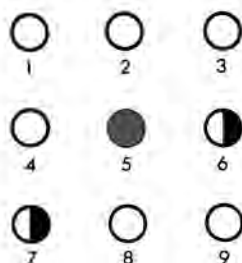


Figura 10.6: Ancora blackjack

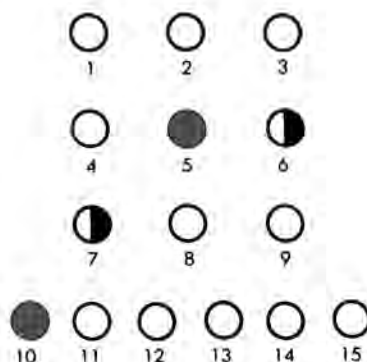


Figura 10.7: Il banco sballa (bust)

ci viene assegnato un 7. Questo era insperabile. Abbiamo fatto ancora tredici!! La situazione è mostrata in Fig. 10.6. È ora il turno del computer. Premiamo il tasto “C”, si accende il LED # 10, il computer ha 15. Il computer è sballato. La situazione è mostrata in Fig. 10.7, verifichiamolo. Premiamo un qualsiasi tasto sulla tastiera. I tre LED di sinistra sulla riga in basso (LED 10, 11 e 12) si accendono e viene visualizzato un punteggio pari a 7. Questo è mostrato in Fig. 10.8. Un istante dopo il visualizzatore si spegne ed inizia una nuova mano.

## IL PROGRAMMA

La Fig. 10.9 riporta un diagramma di flusso dettagliato del programma Blackjack, mentre il listing del programma è riportato alla fine del

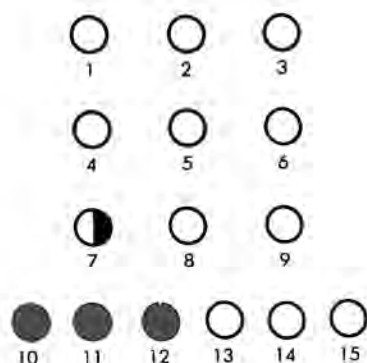


Figura 10.8: Il punteggio finale è 7

capitolo. Come al solito, una parte della pagina 0 è stata riservata alle variabili ed ai flag che non possono essere conservati nei registri interni del 6502. Questa area è mostrata in Fig. 10.10 come “mappa di memoria”. Queste variabili o flag sono:

**DONE:** questo flag è posto al valore “0” all’inizio del gioco. Se il giocatore perde tutto, esso sarà posto al valore “11111111”. Se il giocatore raggiunge 10 punti (il massimo), questo flag sarà posto al valore “1”. Questo flag sarà controllato alla fine del gioco dalla routine ENDER che visualizzerà il risultato finale del gioco sulla scheda ed accenderà una riga di LED oppure un quadrato lampeggiante.

**CHIPS:** questa variabile viene impiegata per memorizzare il punteggio del giocatore. Inizialmente è posta al valore “5”. Ogni volta che il giocatore vince una mano, essa verrà incrementata di uno. Invece, ogni volta che il giocatore perde, essa sarà decrementata di 1. Il gioco termina se questa variabile raggiunge il valore “0” oppure “10”.

**MASKA, MASKB:** queste due variabili vengono impiegate per conservare le maschere o le strutture impiegate per far lampeggiare i LED connessi rispettivamente alla Porta A ed alla Porta B della Scheda Giochi.

**PHAND:** conserva il totale corrente in mano al giocatore. Viene incrementata ogni volta che il giocatore batte (cioè richiede un’altra carta).

**CHAND:** questa variabile conserva il totale corrente in mano al computer (il banco).

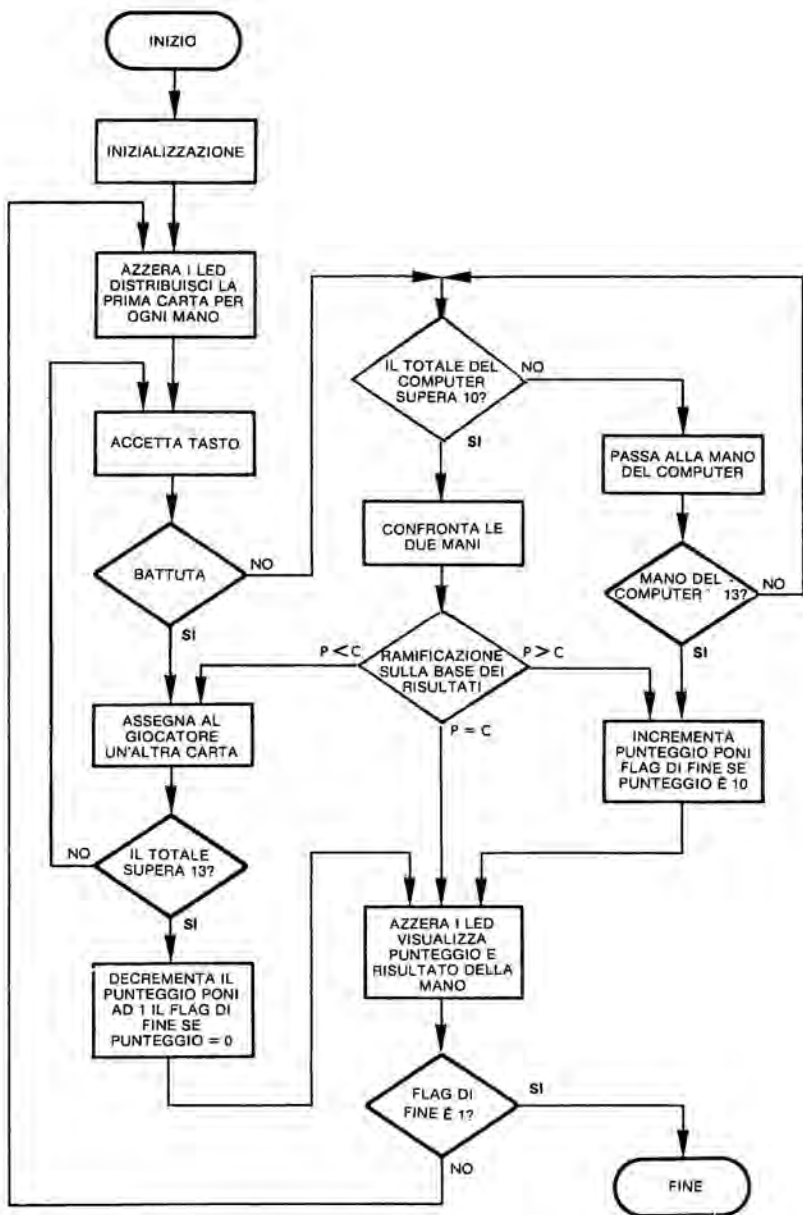


Figura 10.9: Diagramma di flusso del blackjack

TEMP: questa è una variabile temporanea impiegata dalla routine RANDOM per attribuire la carta successiva.

RND fino RND + 5: queste sei locazioni sono riservate alla routine per la generazione di numeri casuali chiamata RANDER.

WHOWON: questo flag di stato viene impiegato per indicare il vincitore della mano. Esso è posto inizialmente a "0" e quindi decrementato se il giocatore perde o se il giocatore vince.

Nella parte alta della memoria, il programma impiega il VIA # 1, la subroutine ACCESS fornita dal SYM monitor ed un vettore di interrupt all'indirizzo A67E, come mostrato in Fig. 10.11.

Esaminiamo ora il funzionamento del programma. Per chiarezza si consiglia di seguire il diagramma di flusso della Fig. 10.9.



Figura 10.10: Mappa dell'area bassa di memoria



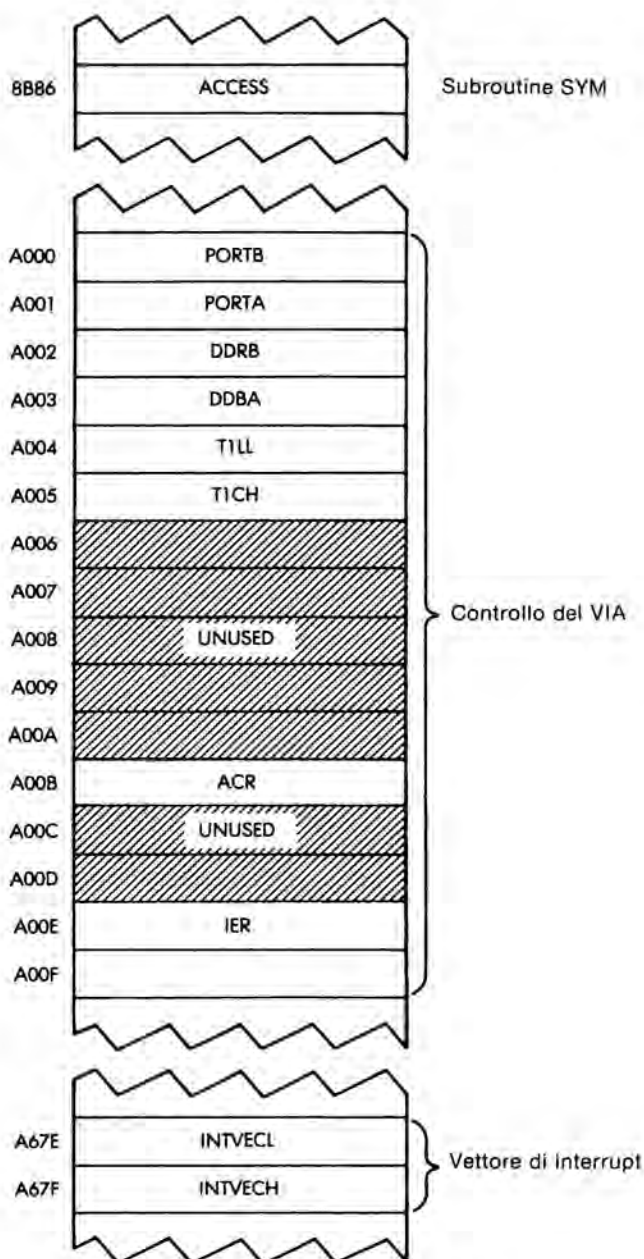


Figura 10.11: Mappa dell'area alta di memoria

## Inizializzazione del programma

Il timer sul 6522 VIA #1 verrà impiegato per generare gli interrupt che fanno lampeggiare i LED. Questi interrupt generano una ramificazione alla locazione 03EA dove c'è la routine di manipolazione interrupt. Il primo passo è quindi quello di caricare il nuovo valore nel vettore di interrupt, cioè "03EA" nella locazione di memoria appropriata:

BLJACK	JSR ACCESS	Sistema di memoria non protetto
	LDA #SEA	Carica il vettore basso di interrupt
	STA INTVECL	
	LDA #\$03	Vettore alto
	STA INTVECH	

Come descritto precedentemente, il registro di abilitazione interrupt viene prima caricato con il valore "01111111" e poi con il valore "11000000" in modo da abilitare l'interrupt per il timer 1:

LDA #\$7F	Azzerà timer abilitazione interrupt
STA IER	
LDA #\$C0	Abilita timer 1 interrupt
STA IER	

Caricando il valore "7F" azzeriamo i bit da 0 a 6, disabilitando quindi tutti gli interrupt. Quindi, caricando il valore "C0", poniamo ad 1 il bit 6, che è il bit di interrupt corrispondente al timer 1. (Vedere Fig. 9.10). Come nel capitolo precedente, il timer 1 viene posto nel modo free-running (a corsa libera). Esso genererà quindi automaticamente interrupt che saranno impiegati per il lampeggio dei LED. Per impostare il modo free-running, il bit 6 di ACR deve essere posto ad "1":

LDA #\$40	Poni il timer 1
STA ACR	Nel modo free-running

I latch del timer 1 sono inizializzati al valore più elevato possibile, cioè FFFF:

LDA #\$FF	
STA TILL	Latch basso del timer 1
STA TICH	Latch alto e start del timer

Infine, ora che il timer è stato correttamente inizializzato, vengono abilitati gli interrupt sul processore:

```
CLI           Abilita gli interrupt
```

Le Porte A e B dei LED sono configurate come uscite (ricordate che l'accumulatore contiene ancora il valore "FF"):

```
STA DDRA
STA DDRB
```

Come precauzione viene azzerato il flag decimale:

```
CLD
```

Il punteggio del giocatore viene inizializzato al valore 5:

```
LDA #5        Poni a 5 il punteggio del giocatore
STA CHIPS
```

Il flag DONE viene inizializzato al valore "0":

```
LDA #0        Azzerare il flag done
STA DONE
```

I LED sulla scheda vengono spenti:

```
STA MASKA
STA MASKB
STA PORTA     Spegni i LED
STA PORTB
```

Ed il flag WHOWON viene anch'esso inizializzato a "0":

```
STA WHOWON Azzerare flag
```

### *Esecuzione della prima mano*

Siamo ora pronti a giocare. Assegniamo una carta al banco ed una al giocatore. Per questo scopo verranno impiegate le subroutine LIGHTR

e BLINKR. Ciascuna di queste subroutine genera un numero casuale ed accende i LED corrispondenti. LIGHTR accende i LED in modo continuo, mentre BLINKR in modo lampeggiante. Queste due subroutine verranno descritte in seguito. Imponiamo il lampeggio di un LED per il giocatore:

JSR BLINKR	Imposta il lampeggio di un LED casuale
------------	--

e salviamo il primo totale della mano in corso del giocatore:

STA PHAND	Memorizza la mano del giocatore
-----------	---------------------------------

quindi eseguiamo la stessa operazione per il computer:

JSR LIGHTR	Accendi fisso un LED casuale
STA CHAND	Memorizza la mano del computer

### *Battuta oppure arresto?*

Ora leggeremo la tastiera. Se il giocatore preme "A" questo indica la richiesta di una carta addizionale, se preme "C" il giocatore passa e viene eseguito il turno del computer. Tutti gli altri tasti vengono ignorati. Leggiamo preliminarmente la chiusura del tasto dalla tastiera:

ASK	JSR GETKEY
-----	------------

Il valore del tasto deve ora venire confrontato con "A" oppure con "C";

CMP #\$0A	
BEQ HITPLR	
CMP #\$0C	È il turno del computer?
BEQ DEALER	

Se è stato premuto qualsiasi altro tasto, verrà ignorato e verrà letto un nuovo tasto:

JMP ASK	Tasto non valido, prova ancora
---------	--------------------------------

A questo punto del programma assumeremo che la situazione richieda una "battuta". Deve essere assegnata un'altra carta al giocatore. Impostiamo un altro LED lampeggiante. Naturalmente le due subroutine BLINKR e LIGHTR, avranno cura di non assegnare una carta già assegnata. Come questo venga ottenuto verrà analizzato in seguito (questo è lo scopo della subroutine SETBIT).

HITPLR	JSR BLINKR	Lampeggio di un LED casuale
--------	------------	-----------------------------

Appena è stata assegnata un'altra carta al giocatore, calcoliamo il nuovo totale del giocatore per la mano in corso:

CLC	
ADC PHAND	Punteggio in mano al giocatore
STA PHAND	

Il nuovo totale deve essere nuovamente controllato con il valore 13. Finchè il giocatore ha 13 o meno, può giocare ancora, cioè decidere se battere o passare. Invece, se il punteggio del giocatore supera 13, egli sballa e perde il gioco. Eseguiamo il controllo:

CMP #14	Controllo per 13
BCC ASK	Chiedi se $\leq 13$
JMP LOSE	Sballa

Ora è il turno del banco. Poichè il computer è molto più veloce del giocatore nel decidere se vuole battere o passare, dobbiamo eseguire un rallentamento per fornire più suspense al gioco:

DEALER	JSR DELAY
--------	-----------

La subroutine di ritardo aumenta il periodo di tempo tra le decisioni successive eseguite dal computer per rendere il computer "simile all'uomo".

Prima di assegnare un'altra carta al computer (il banco), esaminiamo il suo totale. La regola è che il totale del banco non superi "10". (Naturalmente, per gli esperti di Blackjack, sono disponibili altri algoritmi). La mano del computer viene perciò controllata rispetto al valore "10". Se questo valore viene superato, si verifica una ramificazione alla locazione WINNER dove verrà deciso chi vince. Altrimenti viene assegnata una nuova carta al computer:

LDA CHAND

CMP #10

BCS WINNER

Controlla mano per limite

Si. Decide il vincitore.

Finchè il totale della mano è inferiore a "10", il banco ci richiede una battuta. Una nuova carta gli viene assegnata nello stesso modo in cui veniva precedentemente assegnata al giocatore:

JSR LIGHTR

Accendi un LED a caso

Viene calcolato il nuovo totale del banco:

CLC

ADC CHAND

STA CHAND

Punteggio della mano del banco

Come nel caso precedente del giocatore, si confronta con "13" per determinare se il banco è sballato oppure no:

CMP #14

BBC DEALER

JMP WIN

Mano è  $\leq$  13?

Si: altra battuta?

Sballato: vince il giocatore

Se il computer è sballato, si verifica un salto alla locazione WIN, che indica una vittoria da parte del giocatore. Altrimenti si salta indietro alla locazione DEALER, dove il computer determinerà se vuole ricevere una carta addizionale. Determiniamo ora il vincitore:

WINNER

LDACHAND

CMP PHAND

Confronta le mani

Sussistono tre casi possibili: punteggi uguali, il giocatore vince ed il giocatore perde.

BEQ SCORER

BCC WIN

Nel caso che i punteggi siano uguali, si ha un salto alla locazione SCORER che visualizzerà lo stato corrente. Se vince il giocatore si ha un salto alla locazione WIN e la sequenza sarà descritta in seguito. Esaminiamo innanzi tutto cosa succede se il giocatore perde.

### *Il giocatore perde*

Per memorizzare lo stato alla fine di ciascun gioco viene impiegato un flag speciale chiamato WHOWON. Esso viene decrementato per indicare una sconfitta da parte del giocatore:

LOSE            DEC WHOWON

Il punteggio del giocatore viene decrementato:

DEC CHIPS

Il punteggio del giocatore deve essere confrontato con il valore "0". Se tale punteggio ha raggiunto lo zero, il giocatore ha perso. In questo caso il flag DONE è posto a "11111111"; altrimenti rimane invariato. Infine si verifica un salto a SCORER dove verrà visualizzato il punteggio finale:

BNE SCORER	Giocatore al verde?
DEC DONE	Sì: imposta il flag di sconfitta
JMP SCORER	Fine del gioco

### *Il giocatore vince*

Analogamente, quando il giocatore vince, il flag WHOWON viene posto ad "1":

WIN            INC WHOWON

Il punteggio viene incrementato:

INC CHIPS

Esso viene quindi confrontato con il valore "10":

LDA CHIPS	
CMP #10	Gettoni = 10?

Se è stato raggiunto il massimo punteggio pari a "10", viene settato il flag DONE.

BNE SCORER	
INC DONE	Imposta il flag DONE

La visualizzazione dello stato finale viene eseguita dalla routine SCORER. Ricordate che lo stato finale verrà visualizzato soltanto su richiesta del giocatore - quando viene premuto un qualsiasi tasto sulla tastiera. Quindi esiste uno stato di attesa per questo:

```
SCORER    JSR GETKEY
```

Prima di visualizzare lo stato, vengono spenti tutti i LED sulla scheda:

```
LDA #0  
STA MASKA  
STA MASKB  
STA PORTA  
STA PORTB
```

Ora il punteggio del giocatore deve essere visualizzato sulla scheda. Leggiamolo:

```
LDX CHIPS  
BEQ ENDER
```

Se il giocatore non ha altri gettoni, si verifica un salto alla locazione ENDER ed il gioco terminerà. Altrimenti viene visualizzato il punteggio. Sfortunatamente i LED sono numerati internamente da "0" a "7", anche se esternamente sono indicati da "0" ad "8". Per poter accendere il LED appropriato occorre perciò decrementare il punteggio:

```
DEX
```

quindi viene impiegata una subroutine speciale, chiamata SETMASK, per visualizzare il LED appropriato. Si assume che, all'ingresso della subroutine SETMASK, l'accumulatore contenga il numero del LED da visualizzare.

```
TXA  
JSR SETMASK
```

Ora che è stata generata la maschera corretta per visualizzare il punteggio, dobbiamo indicare il vincitore. Se vince il giocatore, vengono accesi i tre LED a sinistra della riga in basso; se vince il computer, vengono accesi i tre LED a destra. Se invece c'è stato un pareggio, nessun LED sarà acceso nella riga in basso. Vediamo chi ha vinto:



```
LDA WHOWON
BEQ ENDER
BMI SC
```

Pareggio: non variare i LED

Se il giocatore ha perso si verifica un salto all'indirizzo SC. D'altra parte, se il giocatore ha vinto, vengono accesi i tre LED a sinistra della riga in basso:

```
LDA #$0E      Ha vinto il giocatore: accendi
               i LED di sinistra
JMP SC0
```

Se il giocatore ha perso, vengono accesi i tre LED di destra:

```
SC      LDA #$B0      Il giocatore ha perso: accendi
               i LED di destra
```

La struttura appropriata per accendere la riga in basso di LED è contenuta nell'accumulatore e questa viene inviata alla Scheda Giochi:

```
SC0      ORA PORTB
          STA PORTB
```

### *Fine del gioco*

La routine ENDER viene impiegata per terminare ciascun gioco. Se il punteggio non è né "0", né "10", viene eseguita una nuova mano:

```
ENDER    JRS DELAY2
          LDA DONE
          BNE EN0
          JMP START
```

Altrimenti, controlliamo il flag DONE per vedere se il giocatore ha vinto oppure ha perso. Se il giocatore ha perso, vengono accesi i LED della riga in basso ed il gioco termina:

```
EN0      BPL EN1      $01:Salta alla condizione
                       di vittoria
          LDA #$BE      Riga fissa di LED
          STA PORTB
          RTS           Ritorno al monitor
```

Nel caso di vittoria del giocatore, viene visualizzato un quadrato lampeggiante ed il programma termina:

```
ENI          LDA #$FF
              STA MASKA
              LDA #$01
              STA MASKB
              RTS
```

## Subroutine

### *Subroutine SETBIT*

Lo scopo di questa subroutine è di creare la struttura richiesta per accendere un dato LED. All'ingresso della subroutine, l'accumulatore contiene un numero compreso tra "0" e "9", che specifica quale LED deve essere acceso. All'uscita della subroutine, il bit corretto è posizionato nell'accumulatore. Se il numero logico del LED è maggiore di "7", il bit carry viene posto ad "1" per indicare che l'uscita dovrebbe verificarsi sulla Porta B, anziché sulla Porta A. Inoltre Y conterrà il valore esterno del LED da accendere (da 1 a 10).

Esaminiamo in dettaglio la subroutine. Il numero del LED viene salvato nel registro indice Y:

```
SETBIT      TAY          Salva il numero logico
```

Esso viene quindi confrontato con il valore limite "7".

```
          CMP #8
          BCC SB0
```

Se il valore era maggiore di 7, sottraiamo 8 da esso:

```
          SBC #8          Sottrai se >7
```

**Esercizio 10.1:** Ricordate che SBC richiede che il carry sia posto ad 1. Siamo in questo caso?

Ora siamo sicuri che il numero contenuto nell'accumulatore è compreso tra "0" e "7". Salviamolo in X:

```
SB0        TAX
```

Un bit sarà ora fatto scorrere nella posizione corrente dell'accumulatore. Preliminarmente poniamo il carry ad "1":

SEC

Prepara la rotazione

Azzeriamo l'accumulatore:

LDA #0

quindi ruotiamo il bit nella posizione corretta:

SBLOOP     ROL A  
             DEX  
             BPL SBLOOP

Notate che il registro indice X viene impiegato come contatore di bit. L'accumulatore è ora condizionato correttamente. Il numero esterno del LED da accendere è uguale al valore inizialmente memorizzato nell'accumulatore più uno:

INY

Rendi Y uguale al # esterno

Se devono essere accesi il LED 9 oppure 10, il bit carry deve essere posto ad 1 per indicare questo fatto. La Porta B dovrà essere impiegata al posto della Porta A:

CPY #9

Poni ad 1 il carry per la  
Porta B

RTS

**Esercizio 10.2:** *Confrontate questa subroutine con la subroutine LIGHT del capitolo precedente.*

**Esercizio 10.3:** *Alla fine, come veniva posto il carry per il LED #9?*

### *Subroutine LIGHTR*

Questa subroutine assegna la carta successiva al banco (computer). Essa genera un numero casuale, si assicura che la carta corrispondente non sia già stata assegnata, cioè che non corrisponda ad una carta già visualizzata sulla scheda. Se non è già stata visualizzata, il numero casuale può essere impiegato come valore della carta successiva da assegnare. Viene quindi acceso un LED nella scheda.

Preliminarmente generiamo un numero casuale:

## LIGHTR JSR RANDOM

Si vedrà in seguito che la routine RANDOM non genera un numero completamente casuale ma è importante assicurarsi che esso non corrisponda ad una carta già assegnata. Quindi occorre posizionare il bit corretto nell'accumulatore per visualizzare la carta. Utilizziamo la routine SETBIT appena descritta, per posizionare il bit nell'accumulatore

### JSR SETBIT

Dobbiamo determinare se deve essere impiegata la Porta A oppure la Porta B. Questo viene eseguito mediante il test del bit carry che è stato condizionato dalla subroutine SETBIT:

### BCS LL0

Assumeremo che debba essere impiegata la Porta A. Il nuovo bit sarà sommato per la visualizzazione mediante OR nella Porta A:

### ORA PORTA STA PORTA

Il valore della carta deve ora essere rimemorizzato nell'accumulatore. Esso è stato salvato nel registro Y dalla routine SETBIT:

### TYA RTS

Nel caso venga impiegata la Porta B, la sequenza è la stessa:

LL0	ORA PORTB	
	STA PORTB	
	TYA	Ripristina valore
	RTS	

### *Subroutine BLINKER*

Questa subroutine funziona esattamente come la precedente LIGHTR tranne per il fatto che i LED vengono fatti lampeggiare. Notate che essa contiene la subroutine SETMASK che imporrà il lampeggio del LED appropriato ed uscirà con il valore numerico del LED nell'accumulatore:

BLINKR	JSR RANDOM	Accetta numero casuale
SETMASK	JSR SETBIT	
	BCS BL0	Salta se Porta B
	ORA MASKA	
	STA MASKA	
	TYA	Ripristina valore
	RTS	
BL0	ORA MASKB	
	STA MASKB	
	TYA	
	RTS	

### *Subroutine RANDOM*

Questa subroutine genererà un numero casuale compreso tra "0" e "9", che non sia già stato impiegato, cioè che non corrisponda al numero interno di un LED già acceso sulla Scheda Giochi. Il valore di questo numero sarà contenuto nell'accumulatore all'uscita della subroutine. Otteniamo un numero casuale:

RANDOM	JSR RANDER	Accetta numero da 0 a 255
--------	------------	---------------------------

La subroutine RANDER è il solito generatore di numeri casuali già descritti ai capitoli precedenti. Come al solito dobbiamo conservare soltanto un numero compreso tra "0" e "9". In questo caso useremo una strategia diversa mediante il rifiuto di qualsiasi numero maggiore di 9 e chiedendo un altro numero se questo si verifica:

```

AND #$0F
CMP #10
BCS RANDOM

```

**Esercizio 10.4:** *Siete in grado di suggerire un metodo alternativo per ottenere un numero compreso tra "0" e "9"? (Suggerimento: un tale metodo è stato descritto nei capitoli precedenti.)*

A questo punto è stato ottenuto un numero casuale compreso tra "0" e "9". Otteniamo ora la corrispondente posizione di bit che deve essere illuminata e salviamola nella locazione di memoria TEMP:

JSR SETBIT	Poni ad "1" il bit corretto
STA TEMP	

Ora controlleremo se il bit corrispondente è già acceso sulla Porta A oppure sulla Porta B. Determiniamo preliminarmente se è la Porta A oppure B:

BCS RN0                      Determina se Porta A oppure B

Assumendo che sia la Porta A, dobbiamo ora trovare quali LED della Porta A sono accesi. Questo viene eseguito combinando le strutture di lampeggio e di accensione continua dei LED, che sono rispettivamente in MASKA ed in PORTA:

LDA MASKA  
ORA PORTA                      Combina PORTA e MASKA

Quindi viene eseguito il controllo per vedere se il bit che si vuole accendere è già acceso:

JMP RN1

Se sì, dobbiamo ottenere un nuovo numero casuale tra "0" e "9":

RN1                      AND TEMP  
                            BNE RANDOM

Se invece il bit non è già acceso, usciamo semplicemente con il valore interno del LED nell'accumulatore:

DEY  
TYA  
RTS

In modo analogo, se deve essere acceso un LED della Porta B, la sequenza è:

RN0                      LDA MASKB  
                            ORA PORTB  
                            AND TEMP  
                            BNE RANDOM  
                            DEY  
                            TYA  
                            RTS

### *Subroutine RANDER*

Questa subroutine genera un numero casuale compreso tra "0" e "255". Essa è già stata descritta ai capitoli precedenti.

### *Subroutine DELAY*

Questo programma impiega due cicli di ritardo: DELAY, che fornisce un ritardo di circa mezzo secondo e DELAY2, che fornisce un ritardo doppio, cioè pari a circa un secondo. I registri indice X ed Y sono entrambi caricati con il valore "FF". Viene quindi implementato il ciclo a due livelli annidati:

```
DELAY2      JSR DELAY
DELAY        LDA #$FF
              TAY
D0           TAX
D1           DEX
              LDA #$FF
              BNE D1
              DEY
              BNE D0
              RTS
```

### **Esercizio 10.5:** *Calcolate la durata esatta delle subroutine DELAY*

#### *Manipolatore di interrupt*

La routine di interrupt viene impiegata per il lampeggio dei LED sulla scheda, mediante MASKA e MASKB, ogni volta che il timer genera un interrupt. Nessun contenuto dei registri viene cambiato. Il funzionamento di questa routine è già stato descritto al capitolo precedente:

```
PHA
LDA PORTA
EOR MASKA
STA PORTA
LDA PORTB
EOR MASKB
STA PORTB
```

## SOMMARIO

Questo programma è notevolmente complesso, nonostante la semplice strategia impiegata dal banco. La maggior parte dei passi logici dell'algoritmo sono accompagnati da effetti sonori e visivi. Notate la piccola quantità di memoria richiesta per l'esecuzione di un gioco relativamente complesso.

**Esercizio 10.6:** *Notate che questo programma assume che i contenuti della locazione di memoria RND siano ragionevolmente casuali all'inizio del gioco. Se voleste avere un valore con maggior grado di casualità in RND all'inizio del gioco, siete in grado di suggerire un'istruzione addizionale da inserire nella fase di inizializzazione di questo programma? (Suggerimento: questo è stato già fatto nei programmi precedenti).*

**Esercizio 10.7:** *Nella routine ENDER, le due istruzioni "BNE EN0" e "JMP START" sono entrambe necessarie? Se non ci fossero, in quali condizioni sarebbero richieste?*

**Esercizio 10.8:** *"Recursione" definisce una routine che chiama se stessa. DELAY2 è recursiva?*



```

; -- BLJACK PROGRAM --
ACCESS = $8B86
INTVECL = $A67E
INTVECH = $A67F
IER = $A00E
ACR = $A00B
T1LL = $A004
T1CH = $A005
DDRA = $A003
DDRB = $A002
PORTA = $A001
PORTB = $A000
MASKA = $C2
MASKB = $C3
CHIPS = $C1
DONE = $C0
PHAND = $C4
CHAND = $C5
TEMP = $C6
RND = $C7
WHOWON = $CD
GETKEY = $100
      = $200

```

; GIOCO BLACKJACK: USA UN 'MAZZO' DI 10 CARTE. LE CARTE ASSEGNATE AL GIOCATORE SONO RAPPRESENTATE DA LED LAMPEGGIANTE QUELLE DEL COMPUTER SONO LED STAZIONARI. LE CARTE SONO ASSEGNATE MEDIANTE UN GENERATORE DI NUMERI CASUALI CHE È NON RIPETITIVO. I TOTALI NUMERICI SONO CONTENUTI NELLE LOCAZIONI DI PAGINA ZERO 'PHAND' E 'CHAND' PORTA E PORTB SONO LE PORTE DEL COMPUTER VERSO IL VISUALIZZATORE A LED. MASKA E MASKB SONO IMPIEGATE DALLA ROUTINE DI INTERRUPT PER IL LAMPEGGIO DEL LED SELEZIONATO 'DONE' E 'WHOWON' SONO I FLAG DI STATO PER DETERMINARE LA FINE DEL GIOCO E CHI VINCE LA MANO CORRENTE.

; IL PROGRAMMA COMINCIA INIZIALIZZANDO IL TIMER ED  
; IL VETTORE DI INTERRUPT. LE PORTE D'USCITA VENGONO  
; ATTIVATE ED I FLAG DI STATO AZZERATI.

```

0200: 20 86 8B BLJACK JSR ACCESS ; MEMORIA DI SISTEMA NON PROTETTA
0203: A9 EA      LDA # $EA      ; CARICA IL VETTORE BASSO DI INTERRUPT
0205: 8D 7E A6      STA INTVECL
0208: A9 03      LDA # $03      ; CARICA IL VETTORE ELEVATO DI
                                ; INTERRUPT
020A: 8D 7F A6      STA INTVECH
020D: A9 7F      LDA # $7F      ; AZZERA IL TIMER DI ABILITAZIONE
                                ; INTERRUPT
020F: 8D 0E A0      STA IER
0212: A9 C0      LDA # $C0      ; ABILITA L'INTERRUPT DEL TIMER 1
0214: 8D 0E A0      STA IER
0217: A9 40      LDA # $40      ; PONI IL TIMER 1 NEL MODO A CORSA
                                ; LIBERA
0219: 8D 0B A0      STA ACR
021C: A9 FF      LDA # $FF
021E: 8D 04 A0      STA T1LL
0221: 8D 05 A0      STA T1CH
0224: 58          CLI          ; PONI LATCH BASSO SUL TIMER 1
                                ; PONI LATCH ALTO & AVVIO DEL TIMER
                                ; ABILITA GLI INTERRUPT DEL
                                ; PROCESSORE
0225: 8D 03 A0      STA DDRA    ; CONFIGURA LE PORTE DEI LED COME
                                ; USCITE
0228: 8D 02 A0      STA DDRB

```

Figura 10.12: Programma del blackjack.

```

022B: D8          CLD
022C: A9 05        LDA #5      ; PONI A 5 IL PUNTEGGIO DEL GIOCATORE
022E: 85 C1        STA CHIPS
0230: A9 00        LDA #0      ; AZZERA IL FLAG DONE
0232: 85 C0        STA DONE

;
; NUOVA MANO: IL DISPLAY È AZZERATO. SONO IMPOSTATI
; I VALORI INIZIALI DI ENTRAMBE LE MANI ED I LED
; CORRISPONDENTI.
;
0234: 85 C2        START STA MASKA ; AZZERA LA MASCHERA DI LAMPEGGIO; SI
; ASSUME
0236: 85 C3          STA MASKB ; CHE L'ACCUMULATORE CONTENGA ZERO
0238: 8D 01 A0      STA PORTA ; AZZERA I LED
023B: 8D 00 A0      STA PORTB
023E: 85 CD        STA WHOWON ; AZZERA IL FLAG DELLA MANO
0240: 20 0F 03      JSR BLINKR ; IMPOSTA IL LAMPEGGIO DI UN LED
; CASUALE
0243: 85 C4          STA PHAND ; MEMORIZZA LA MANO DEL GIOCATORE
0245: 20 F7 02      JSR LIGHTR ; IMPOSTA UN LED CASUALE STAZIONARIO
0248: 85 C5          STA CHAND ; MEMORIZZA LA MANO DEL COMPUTER

;
; INGRESSO DI TASTO: 'A' È UNA BATTUTA. 'C' È IL TURNO DEL COM-
; PUTER E TUTTI GLI ALTRI TASTI VENGONO IGNORATI
;
024A: 20 00 01      ASK JSR GETKEY ; ACCETTA UN NUOVO INGRESSO
024D: C9 0A          CMP # $0A ; IL GIOCATORE ESEGUE UNA BATTUTA?
024F: F0 07          BEQ HITPLR ; SÌ, SALTA
0251: C9 0C          CMP # $0C ; È IL TASTO 'TURNO DEL COMPUTER'?
0253: F0 12          BEQ DEALER ; SÌ
0255: 4C 4A 02      JMP ASK ; TASTO ERRATO, PROVA ANCORA

0258: 20 0F 03      HITPLR JSR BLINKR ; IMPOSTA UN TEST CASUALE
025B: 18            CLC
025C: 65 C4          ADC PHAND ; AVANZAMENTO MANO DEL COMPUTER
025E: 85 C4          STA PHAND
0260: C9 0E          CMP #14 ; CONTROLLA MANO
0262: 90 E6          BCC ASK ; SE <= 13, OK
0264: 4C B7 02      JMP LOSE ; SBALLATO, BUSTED, VIA ALLA ROUTINE
; DI SCONFITTA

0267: 20 5D 03      DEALER JSR DELAY ; ESECUZIONE DEL RITARDO DELLA
; ROUTINE
026A: A5 C5          LDA CHAND ; IL COMPUTER È FUORI LIMITE?
026C: C9 0A          CMP #10
026E: B0 0F          BCS WINNER ; SÌ, RAPPRESENTA IL VINCITORE
0270: 20 F7 02      JSR LIGHTR ; NO, IMPOSTA UN LED CASUALE
0273: 18            CLC
0274: 65 C5          ADC CHAND ; AVANZAMENTO MANO DEL COMPUTER
0276: 85 C5          STA CHAND
0278: C9 0E          CMP #14 ; MANO È <= 13?
027A: 90 EB          BCC DEALER ; SÌ, UN'ALTRA BATTUTA?
027C: 4C 92 02      JMP WIN ; BUSTED, VINCE IL GIOCATORE

;
; DETERMINA IL VINCITORE: AVANZA IL PUNTEGGIO DI VITTORIA E
; SCONFITTA. DETERMINARE SE IL GIOCATORE HA VINTO O HA PERSO
; IL GIOCO. VIENE IMPOSTATO IL FLAG "WOWON" PER INDICARE CHI
; VINCE LA PARTICOLARE MANO. SE LE MANI SONO UGUALI, NON
; CAMBIA NULLA.
;
027F: A5 C5        WINNER LDA CHAND ; CONFRONTA LE MANI

```

Figura 10.12: Programma del blackjack (continua)

```

0281: C5 C4      CMP PHAND
0283: F0 19      BEQ SCORER      ; SONO UGUALI, NESSUNA VARIAZIONE
0285: 90 0B      BCC WIN        ; LA MANO DEL GIOCATORE È MAGGIORE
0287: C6 CD      LOSE      DEC WHOWON    ; ROUTINE DI SCONFITTA
0289: C6 C1      DEC CHIPS      ; AVANZAMENTO PUNTEGGIO
028B: D0 11      BNE SCORER      ; IL GIOCATORE HA FINITO I GETTONI?
028D: C6 C0      DEC DONE      ; SÌ, IMPOSTA AD 1 IL FLAG DI FINE
                                ; GIOCO: SCONFITTA

028F: 4C 9E 02   JMP SCORER
0292: E6 CD      WIN      INC WHOWON    ; ROUTINE DI VITTORIA
0294: E6 C1      INC CHIPS      ; AVANZA IL PUNTEGGIO
0296: A5 C1      LDA CHIPS      ; SOMMA LE VINCITE
0298: C9 0A      CMP #10      ; SE GETTONI = 10, PONI AD 1 IL FLAG DI
                                ; FINE GIOCO

029A: D0 02      BNE SCORER
029C: E6 C0      INC DONE      ; PONI AD 1 IL FLAG DI FINE GIOCO:
                                ; VITTORIA

                                ;
                                ; VISUALIZZA IL PUNTEGGIO ACCENDENDO UNO DEI 10 LED. LA RIGA
                                ; IN BASSO DI LED MOSTRA CHI VINCE LA MANO. VIENE ESEGUITO UN
                                ; TEST DI FINE GIOCO, IN CASO AFFERMATIVO, VENGONO MOSTRATI
                                ; I LED ED IL PROGRAMMA TERMINA. SI ASSUME CHE L'INDIRIZZO DEL
                                ; MONITOR SIA NELLO STACK.
                                ;
029E: 20 00 01   SCORER JSR GETKEY      ; CONSERVA L'ULTIMA SITUAZIONE
                                ; DELLE CARTE
02A1: A9 00      LDA #0              ; AZZERA I LED
02A3: 85 C2      STA MASKA
02A5: 85 C3      STA MASKB
02A7: 8D 01 A0   STA PORTA
02AA: 8D 00 A0   STA PORTB
02AD: A6 C1      LDX CHIPS      ; VISUALIZZA IL NUMERO DI GETTONI
02AF: F0 18      BEQ ENDER      ; AGGIUSTA IN MODO CHE LA SUBROUTINE
                                ; POSSA
                                ; IMPOSTARE IL LED CORRETTO

02B1: CA        DEX
02B2: 8A        TXA
02B3: 20 12 03   JSR SETMASK

02B6: A5 CD      LDA WHOWON      ; VEDI CHI VINCE LA MANO
02B8: F0 0F      BEQ ENDER      ; PAREGGIO - NON VARIARE I LED
02BA: 30 05      BMI SC
02BC: A9 0E      LDA #$0E      ; VINCE IL GIOCATORE - ACCENDI I TRE
                                ; LED DI SINISTRA

02BE: 4C C3 02   JMP SC0
02C1: A9 B0      SC      LDA #$B0      ; IL GIOCATORE PERDE - ACCENDI I TRE
                                ; LED DI DESTRA
02C3: 0D 00 A0   SC0      ORA PORTB      ; IMPOSTA LA PORTA DEI LED
02C6: 8D 00 A0   STA PORTB
02C9: 20 5A 03   ENDER      JSR DELAY2      ; CONSERVA LA VISUALIZZAZIONE

02CC: A5 C0      LDA DONE      ; CONTROLLA LA CONDIZIONE DI FINE
                                ; GIOCO

02CE: D0 03      BNE EN0
02D0: 4C 34 02   JMP START
02D3: 10 06      EN0      BPL EN1      ; ZERO, INIZIO NUOVA MANO
02D5: A9 BE      LDA #$BE      ; $01, CONDIZIONE VINCENTE
02D7: 8D 00 A0   STA PORTB      ; IMPOSTA UNA RIGA PIENA DI LED
02DA: 60        RTS          ; RITORNO AL MONITOR

02DB: A9 FF      EN1      LDA #$FF      ; IMPOSTA IL QUADRATO DI LAMPEGGIO
02DD: 85 C2      STA MASKA

```

Figura 10.12: Programma del blackjack (continua)

```

02DF: A9 01      LDA # $01
02E1: 85 C3      STA MASKB
02E3: 60          RTS          ; RITORNO AL MONITOR

;
;
; -SUBROUTINE-
;
; PONI UN BIT NELL'ACCUMULATORE: ENTRA CON UN VALORE LOGICO
; I.E. 0-9 NELL'ACC. ESCI CON UN VALORE NUMERICO (1-10) IN Y, ED IL
; BIT POSIZIONATO NELL'ACCUMULATORE

02E4: A8          SETBIT TAY          ; SALVA IL NUMERO LOGICO
02E5: C9 08      CMP #8             ; DELIMITA IL VALORE 0-7
02E7: 90 02      BCC SB0
02E9: E9 08      SBC #8
02EB: AA          SB0 TAX             ; ... SOTTRAI SE : 7
02EC: 38          SEC             ; IMPOSTA IL REGISTRO INDICE
02ED: A9 00      LDA #0            ; PREPARA IL BIT ALLA ROTAZIONE
02EF: 2A          SBLOOP ROL A      ; MUOVI IL BIT IN POSIZIONE
02F1: 10 FC      BPL SBLOOP
02F3: C8          INY             ; RENDI Y NUMERICO, NON LOGICO
02F4: C0 09      CPY #9            ; PONI AD 1 IL CARRY, PER LA PORTA B, C=1
02F6: 60          RTS

;
; LIGHTR: ACCENDE UN LED CASUALE STAZIONARIO CHE NON
; SIA GIA' ACCESSO UN NUMERO CASUALE, QUINDI PONE IL BIT NELLA
; PORTA CORRETTA, IL VALORE NUMERICO DEL BIT VIENE POSTO
; NELL'ACCUMULATORE ALL'ATTO DELL'USCITA

02F7: 20 23 03 LIGHTR JSR RANDOM    ; ACCETTA NUMERO CASUALE
02FA: 20 E4 02 JSR SETBIT          ; ACCETTA IL BIT POSIZIONATO
; NELL' ACC.
02FD: B0 08      BCS LL0           ; SALTA SE PREDISPOSTA PORTA B
02FF: 0D 01 A0   ORA PORTA         ; ACCENDI IL LED NELLA PORTA A
0302: 8D 01 A0   STA PORTA
0305: 98          TYA             ; RIPRISTINA IL VALORE NUMERICO
0306: 60          RTS
0307: 0D 00 A0 LL0 ORA PORTB       ; ACCENDI IL LED NELLA PORTA B
030A: 8D 00 A0   STA PORTB
030D: 98          TYA             ; RIMEMORIZZA VALORE NUMERICO
030E: 60          RTS

;
; BLINKER: ACCENDE IN MODO LAMPEGGIANTE UN LED PRECEDENTE-
; MENTE SPENTO ALL'USCITA IL VALORE NUMERICO DEL LED E' NEL-
; L'ACC. ACCETTA UN NUMERO CASUALE, QUINDI LO FORNISCE ALLA
; ROUTINE SETMASK PER LAMPEGGIARE IL LED APPROPRIATO.
; SETMASK: ENTRA CON UN VALORE LOGICO E LA ROUTINE ACCENDE
; IN MODO LAMPEGGIANTE IL LED CORRETTO. SI ESCE CON IL VALORE
; NUMERICO DEL LED DELL'ACCUMULATORE.

030F: 20 23 03 BLINKR JSR RANDOM    ; ACCETTA IL NUMERO CASUALE
0312: 20 E4 02 SETMASK LSR SETBIT
0315: B0 06      BCS BL0           ; SE PREDISPOSTA PORTA B
0317: 05 C2      ORA MASKA
0319: 85 C2      STA MASKA

```

Figura 10.12: Programma del blackjack (continua)

```

031B: 98          TYA          ; RIMEMORIZZA IL VALORE NUMERICO
031C: 60          RTS
031D: 05 C3      BLQ          ; PONI MASKB
031F: 85 C3      STA MASKB
0321: 98          TYA
0322: 60          RTS

; GENERA UN NUMERO CASUALE DA 0 A 9 CHE NON SIA IL NUMERO
; DI UN LED GIA' ACCESO. ALL'USCITA IL RISULTATO E
; NELL'ACCUMULATORE

0323: 20 47 03   RANDOMJSR RANDE ; ACCETTA NUMERO DA 0 A 255
0326: 29 0F      AND # $0F      ; MASCHERA IL NIBBLE DI ORDINE
; ELEVATO
; LIMITA DA 0 A 9
0328: C9 0A      CMP #10
032A: B0 F7      BCS RANDOM
032C: 20 E4 02   JSR SETBIT    ; POSIZIONA IL BIT
032F: 85 C6      STA TEMP      ; SALVALO
0331: B0 08      BCS RN0       ; DETERMINA PORT A OPPURE B
0333: A5 C2      LDA MASKA     ; COMBINA PORT E MASK
0335: 0D 01 A0   ORA PORTA
0338: 4C 40 03   JMP RN1
033B: A5 C3      RN0 LDA MASKB  ; COMBINA PORT E MASK
033D: 0D 00 A0   ORA PORTB
0340: 25 C6      RN1 AND TEMP  ; OSSERVA IL BIT SPECIFICO
0342: D0 DF      BNE RANDOM    ; SE BIT GIA' USATO, PROVA ANCORA
0344: 88          DEY          ; RENDI Y LOGICO
0345: 98          TYA          ; ESCI CON VALORE IN ACCUMULATORE
0346: 60          RTS

; GENERA UN NUMERO CASUALE DA 0 A 255. USA I NUMERI A, B, C, D, E,
; F
; MEMORIZZATI DA RND AD RND+5. SOMMA B+E+F+1 E PONI IL
; RISULTATO IN A, QUINDI FA SCORRERE A IN B, B IN C, ECC.
; ALL'USCITA IL NUMERO CASUALE E NELL'ACCUMULATORE.

0347: 38          RANDE SEC    ; CARRY AGGIUNGE 1
0348: A5 C8      LDA RND+1    ; SOMMA B, D, F
034A: 65 CB      ADC RND+1
034C: 65 CC      ADC RND+5
034E: 85 C7      STA RND
0350: A2 04      LDX #4       ; SCORRIMENTO IN AVANTI DEI NUMERI
0352: B5 C7      RDLOOP LDA RND,X
0354: 95 C8      STA RND+1,X
0356: CA          DEX
0357: 10 F9      BPL RDLOOP
0359: 60          RTS

; CICLO DI RITARDO: IL RITARDO DI DELAY2 E' SEMPLICEMENTE DOPPIO
; DI DELAY. IL CICLO ASSEGNATO HA UN RITARDO DI CIRCA 0,5 SEC.

035A: 20 5D 03   DELAY2 JSR DELAY
035D: A9 FF      DELAY LDA # $FF ; IMPOSTA IL VALORE PER I CICLI
035F: A8          TAY
0360: AA          D0 TAX
0361: CA          D1 DEX
0362: A9 FF      LDA # $FF
0364: D0 FB      BNE D1
0366: 88          DEY
0367: D0 F7      BNE D0
0369: 60          RTS

```

Figura 10.12: Programma del blackjack (continua)

ROUTINE DI INTERRUPT: OR ESCLUSIVO DELLE PORTE D'USCITA  
 CON LE CORRISPONDENTI MASCHERE DI LAMPEGGIO OGNI VOLTA  
 CHE  
 SI HA IL TIME OUT DEL TIMER PER LAMPEGGIARE I LED SELEZIONATI.  
 NESSUN REGISTRO VIENE CAMBIATO ED IL FLAG DI INTERRUPT  
 VIENE AZZERATO PRIMA DELL'USCITA.

```

      = $03EA
03EA: 48          PHA          ; SALVA L'ACCUMULATORE
03EB: AD 01 A0    LDA PORTA    ; COMPLEMENTA LE PORTE CON LE
                                ; MASCHERE
03EE: 45 C2      EOR MASKA
03F0: 8D 01 A0    STA PORTA
03F3: AD 00 A0    LDA PORTB
03F6: 45 C3      EOR MASKB
03F8: 8D 00 A0    STA PORTB
03FB: AD 04 A0    LDA T1LL     ; AZZERA IL BIT DI INTERRUPT DEL TIMER
03FE: 68          PLA          ; RIMEMORIZZA L'ACCUMULATORE
03FF: 40          RTI
  
```

#### SYMBOL TABLE

ACCESS	8B86	INTVECL	A67E	INTVECH	A67F
IER	A00E	ACR	A00B	T1LL	A004
T1CH	A005	DDRA	A003	DDRB	A002
PORTA	A001	PORTB	A000	MASKA	00C2
MASKB	00C3	CHIPS	00C1	DONE	00C0
PHAND	00C4	CHAND	00C5	TEMP	00C6
RND	00C7	WHOWON	00CD	GETKEY	0100
BLJACK	0200	START	0234	ASK	024A
HITPLR	0258	DEALER	0267	WINNER	027F
LOSE	0287	WIN	0292	SCORER	029E
SC	02C1	SC0	02C3	ENDER	02C9
EN0	02D3	EN1	02DB	SETBIT	02E4
SB0	02EB	SBLOOP	02EF	LIGHTR	02F7
LL0	0307	BLINKR	030F	SETMASK	0312
BL0	031D	RANDOM	0323	RN0	033B
RN1	0340	RANDER	0347	RDLOOP	0352
DELAY2	035A	DELAY	035D	D0	0360
D1	0361				

Figura 10.12: Programma del blackjack

## CAPITOLO 11

# TIC-TAC-TOE

### LE REGOLE

Il Tic-Tac-Toe è un gioco che viene eseguito su un quadrato di tre per tre caselle. Il simbolo a disposizione del giocatore è una "O", mentre per il computer si usa una "X". I giocatori muovono a turno, il primo giocatore che riesce ad allineare tre simboli (orizzontale, verticale oppure diagonale) è il vincitore del gioco. La Fig. 11.1 riporta una delle otto possibili combinazioni vincenti. Un LED acceso fisso sarà impiegato per visualizzare una "X", cioè la mossa del computer, mentre un LED lampeggiante, visualizzerà una "O", cioè la mossa del giocatore.

Possono fare la prima mossa sia il giocatore che il computer. Se il giocatore decide di muovere per primo, deve premere il tasto "F". Se deve muovere per primo il computer, deve essere premuto un qualsiasi altro tasto perchè il computer stesso inizi il gioco. Alla fine di ciascun gioco si avrà uno start automatico. Il computer è dotato di una variabile IQ (intelligenza) il cui valore va da uno a quindici. Ogni volta che il computer vince, il suo livello IQ viene ridotto di un'unità. Ogni volta che vince il giocatore, il livello IQ del computer viene aumentato di un'unità. In questo modo ogni giocatore ha la possibilità di vincere. Ogni volta che vince il giocatore viene emesso un suono acuto, ogni volta che perde ne viene emesso uno basso.

### UNA GIOCATA TIPICA

Il display è inizialmente spento. Lasciamo la partenza al computer. Imponiamo questo premendo un tasto qualsiasi, eccetto il tasto "F". (Se premiamo il tasto "F", allora la prima mossa tocca al giocatore.) Cominciamo premendo "O". Dopo una breve pausa il computer risponde con un "chirp" (cinguettio) ed esegue la sua mossa. (Vedere Fig. 11.2)

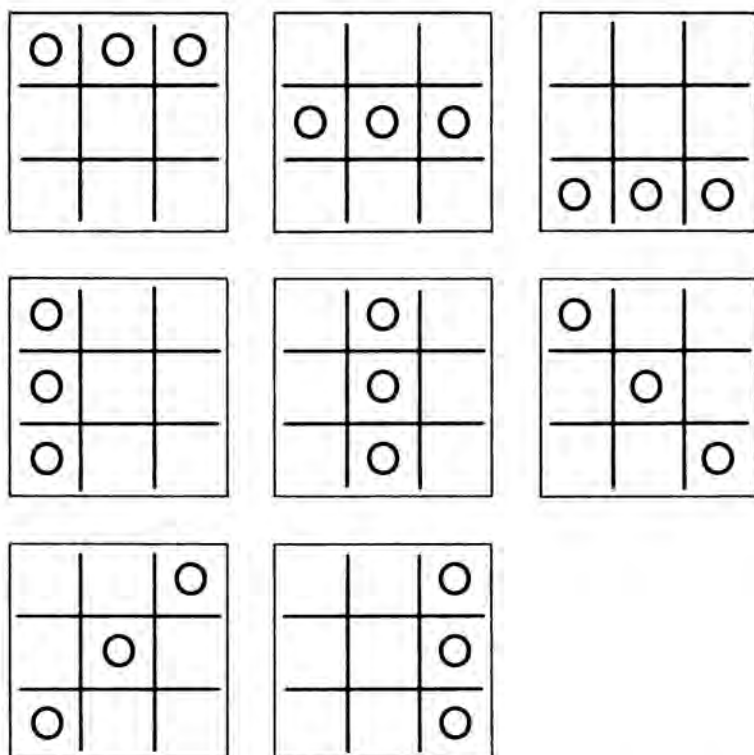


Figura 11.1: Combinazioni vincenti per un giocatore al tic-tac-toe

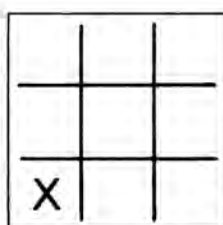


Figura 11.2: Prima mossa del computer

Una "X" viene impiegata per indicare le mosse del computer, una "O" denoterà le nostre mosse. Nelle figure mostrate dei LED spenti si useranno degli spazi bianchi.



Ci muoviamo al centro ed occupiamo la posizione 5. (Vedere Fig. 11.3). Premiamo il tasto "5". L'istante successivo si accende il LED # 1 e viene emesso un chirp per indicare che è di nuovo il nostro turno. La scheda è mostrata in Fig. 11.4.

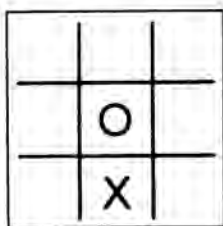


Figura 11.3: La nostra prima mossa

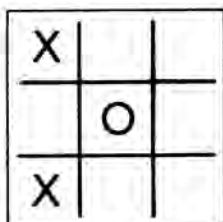


Figura 11.4: Seconda mossa del computer

È ora il nostro turno e dovremo bloccare il computer per prevenire il completamento di una colonna vincente. Occupiamo la posizione 4. Premiamo il tasto "4". L'istante successivo si accende il LED #6 e viene emesso un chirp. La situazione è mostrata in Fig. 11.5.

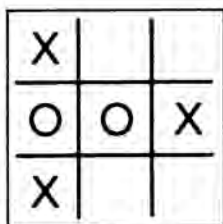


Figura 11.5: Dopo la terza mossa del computer

Occupiamo la posizione 2. Il computer risponde occupando la posizione 8 (Fig. 11.6) Preveniamo il computer dal completare una riga vincente occupando la posizione 9. Il computer risponde occupando la posizione 3 (Fig. 11.7). È una situazione di pareggio. Nessuno vince, tutti i LED sulla scheda lampeggiano per un momento e poi si spengono. Si può iniziare con un altro gioco.

X	O	
O	O	X
X	X	

Figura 11.6: Dopo la quarta mossa del computer

X	O	X
O	O	X
X	X	O

(PAREGGIO)

Figura 11.7: Dopo la quinta mossa del computer

## Un altro gioco

Questa volta vogliamo partire e, possibilmente, vincere ! Premiamo "F" per iniziare il gioco. Viene emesso un chirp per conferma che è il nostro turno. Giochiamo nella posizione 5. Il computer risponde occupando la posizione 3. Viene emesso un chirp per avvisarci che dobbiamo giocare ancora. La situazione è mostrata in Fig. 11.8. Occupiamo la posizione 4. Il computer risponde occupando il quadrato 6 (Fig. 11.9). Questa volta dobbiamo bloccare il computer dal completare la colonna di destra, occupiamo la posizione 9. Il computer risponde muovendo nel

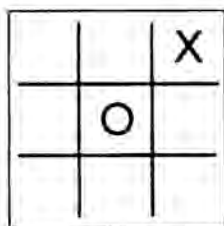


Figura 11.8: Mossa 1

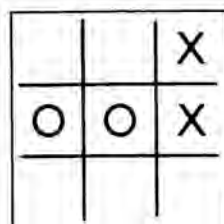


Figura 11.9: Mossa 2

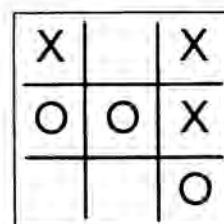


Figura 11.10: Mossa 3

quadrato 1, impedendoci di completare una diagonale. Questa situazione è mostrata in Fig. 11.10. Dobbiamo impedire al computer di completare la riga in alto; quindi occupiamo la posizione 2. Il computer risponde occupando la posizione 8. Questo è mostrato in Fig. 11.11. Facciamo la mossa che termina il gioco nel quadrato 7. Si tratta di un pareggio; non siamo riusciti a battere il computer.

Poichè il computer è “abbastanza astuto” da occupare la posizione sulle diagonali dopo che abbiamo occupato la posizione centrale, non

X	O	X
O	O	X
	X	O

Figura 11.11: Mossa 4

siamo riusciti a vincere. Nota: se ripetiamo più volte il tentativo, qualche volta il computer occuperà una delle posizioni laterali (2, 4, 6 oppure 8) anziché una delle posizioni d'angolo ed in questo caso avremo la possibilità di vincere. Ecco un esempio.

Occupiamo la posizione centrale. Il computer risponde muovendo nella posizione 6. La situazione è rappresentata in Fig. 11.12. Occupiamo il quadrato 1; il computer muove nel quadrato 9 (Fig. 11.13). Muoviamo al quadrato 3; il computer muove al quadrato 7. (Fig. 11.14).

	O	X

Figura 11.12: Mossa 1

O		
	O	X
		X

Figura 11.13: Mossa 2

Questa volta abbiamo la mossa vincente giocando nel quadrato 2. La situazione è mostrata in Fig. 11.15. Notate che, se iniziamo il gioco e se giochiamo bene, il risultato può essere un pareggio oppure una vittoria. Nel gioco del Tic-Tac-Toe, chi inizia non può perdere se non commette errori grossolani.

O		O
	O	X
X		X

Figura 11.14: Mossa 3

O	O	O
	O	X
X		X

Figura 11.15: "Abbiamo vinto!"

## L'ALGORITMO

L'algoritmo del programma Tic-Tac-Toe è il più complesso di quelli progettati fino ad ora. Esso appartiene al dominio della cosiddetta "intelligenza artificiale". Questo termine viene impiegato per denotare il fatto che le funzioni eseguite dal programma sono una duplicazione dell'attività mentale comunemente chiamata "intelligenza". La progettazione di un buon algoritmo per questo gioco con l'occupazione di un piccolo spazio di memoria non è un problema banale. Storicamente sono stati proposti numerosi algoritmi e se ne possono ottenere molti altri. In questa sede esamineremo in dettaglio due strategie e quindi ne selezioneremo ed implementeremo una. Sugeriremo degli esercizi addizionali per altre possibili strategie.

## Strategia per decidere la mossa successiva

Per determinare la mossa successiva eseguita dal computer possono essere utilizzate numerose strategie. L'approccio più diretto potrebbe essere quello di memorizzare tutte le possibili configurazioni ed esaminare il miglior responso. Questo è il miglior metodo da impiegare da un punto di vista matematico dell'analisi in quanto la miglior mossa possibile è sicuramente quella eseguita di volta in volta. In questo caso si tratta anche di un approccio, in quanto il numero di combinazioni in una matrice  $3 \times 3$  è limitato. Comunque, avendo già imparato in altri giochi l'esecuzione di tabelle di consultazione, un tale approccio non potrebbe insegnarci molto in materia di programmazione. Esso potrebbe anche essere considerato "non onesto". Quindi analizzeremo altri metodi applicabili ad una vasta gamma di giochi, oppure ad una matrice più grande.

Si possono proporre molte strategie. Per esempio, è possibile considerare una strategia *euristica* nella quale il computer *impara eseguendo*. In altre parole il computer migliora la sua tattica di gioco all'aumentare dei giochi eseguiti imparando dagli errori commessi. Questa è la strategia in cui le mosse iniziali eseguite dal computer sono casuali. Comunque, supposto che sia disponibile una quantità di memoria sufficiente, il computer ricorderà ogni mossa che è stata eseguita. Se esso è incappato in una sconfitta, le mosse che lo portano ad essa saranno respinte dal computer e considerate come mosse non giuste, quindi non saranno più impiegate nella stessa sequenza. Con il tempo e con un algoritmo "di apprendimento" ragionevole questo approccio si risolve nella costruzione di *tabelle di decisione*. Comunque, questo approccio assume che sia disponibile una grande quantità di memoria. Questo non è il nostro caso. Vogliamo progettare un programma che risieda in circa 1 K di memoria. Consideriamo un altro approccio.

Un altro approccio fondamentale consiste nella *valutazione della matrice* dopo ogni mossa. La matrice dovrebbe essere esaminata da due punti di vista: primo, se ci sono due "O" allineate, è importante bloccarle per evitare che possa essere ottenuta una vittoria con una mossa successiva. Inoltre ogni volta si dovrebbe esaminare la configurazione della matrice in funzione di una *vittoria potenziale*: per esempio, se due "X" sono in una riga il programma deve muovere in modo da completare la riga ottenendo una vittoria. Naturalmente queste due situazioni sono molto semplici da rivelare. Il problema reale risiede nella valutazione potenziale di ogni quadrato della matrice in ogni situazione.

## Un algoritmo analitico

A questo punto, mostreremo il processo impiegato per progettare un

algoritmo guidato da principi molto generali. Successivamente scopriremo la debolezza dell'algoritmo e lo miglioreremo. Questo ci servirà come esempio di un possibile approccio alla risoluzione del problema in una strategia di gioco.

### *Concetto generale*

Il concetto fondamentale è di valutare il potenziale di ogni quadrato della matrice da due punti di vista: "vittoria" e "minaccia". Il *potenziale di vittoria* corrisponde alla possibilità di vincere giocando con un particolare quadrato. Il *potenziale di minaccia* è il potenziale di vittoria dell'avversario.

Dobbiamo preliminarmente decidere un modo per assegnare un valore numerico ad una combinazione di "O" ed "X" sulla matrice. Questo deve essere eseguito in modo da poter calcolare il valore strategico o "potenziale" di un quadrato della matrice.

### *Calcolo del valore*

Per ogni riga (o colonna oppure diagonale) si possono verificare quattro configurazioni: -escludendo il caso in cui tutte e tre le posizioni siano già occupate e non possiamo eseguire una mossa su quella riga. Queste configurazioni sono mostrate in Fig. 11.16. La situazione "A" corrisponde al caso in cui tutti i tre quadrati sono vuoti. Chiaramente la situazione ha qualche possibilità ed assegnamo il valore "uno" ad ogni quadrato in questo caso. Il caso successivo è rappresentato nella riga "B" della Figura 11.16; esso corrisponde alla situazione in cui c'è già una "X" in quella riga. Se poniamo una seconda "X" in quella riga, potremmo essere molto vicini ad una vittoria. Questa è una situazione auspicabile che ha un valore maggiore dell'uno precedente. Quindi sommiamo "uno" al valore del quadrato libero per la presenza della "X"; il valore di ogni quadrato in questo caso sarà "due".

Consideriamo il caso "C" della Figura 11.16. nella quale abbiamo una "X" ed una "O". La configurazione non ha alcun valore perchè non saremo in grado di vincere in quella particolare riga. La presenza di una "O" porta il valore del quadrato rimanente a "zero".

Infine esaminiamo la situazione della riga "D" della Fig. 11.16, dove ci sono già due "X". Chiaramente questa è una situazione vincente dove si dovrebbe avere il valore più elevato. Assegnamole il valore "tre".

Il concetto successivo è che ogni quadrato sulla matrice appartiene ad una riga, una colonna e, possibilmente, ad una diagonale. Ogni quadrato dovrebbe perciò essere valutato in due o tre direzioni. Eseguiremo

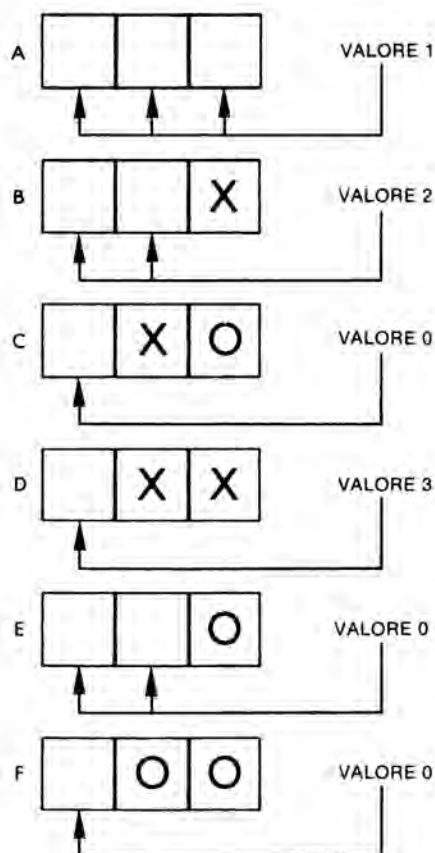


Figura 11.16: Le sei combinazioni

H	V			
D	T			

Figura 11.17: Reticolo di valutazione



questo e quindi totalizzeremo i potenziali di ogni direzione. Per semplicità utilizzeremo un reticolo di valutazione come mostrato in Fig. 11.17. Ogni quadrato di questo reticolo è stato suddiviso in quattro più piccoli. Questi quadrati interni vengono impiegati per visualizzare il potenziale di ciascun quadrato in ogni direzione. Il quadrato contrassegnato con "H" nella Fig. 11.17 verrà impiegato per valutare il potenziale di riga orizzontale. "V" sarà impiegato per il potenziale di colonna verticale mentre "D" verrà impiegato per il potenziale di diagonale. "T" verrà impiegato per il totale dei tre precedenti quadrati. Notate che non esiste un valore diagonale per quattro quadrati sulla matrice. Questo perché essi non si trovano su diagonalì. Inoltre notate che il quadrato al centro ha due valori diagonali in quanto si trova all'intersezione delle due diagonalì.

Una volta che il nostro algoritmo ha calcolato i potenziali di vittoria e di minaccia totali, deve decidere qual'è il miglior quadrato in cui effettuare una mossa. La soluzione ovvia è quella di muovere al quadrato avente il più alto potenziale di vittoria o di minaccia.

Ora determineremo il valore del nostro algoritmo in alcuni casi reali. Considereremo alcune configurazioni tipiche della matrice e le valuteremo impiegando i nostri algoritmi per valutare se le mosse che essi generano hanno senso.

### *Un test dell'algoritmo iniziale*

Osserviamo la Fig. 11.18. In questa situazione è il turno del giocatore ("O"). Valuteremo la matrice da due punti di vista: potenziale per "X" e minaccia per "O". Quindi selezioneremo il quadrato avente il totale più elevato nei due reticoli generati ed in questa posizione eseguiremo la nostra mossa.

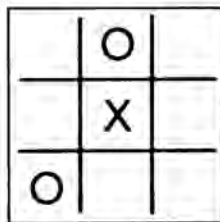


Figura 11.18: Caso di prova 1

Preliminarmente completiamo il reticolo di valutazione della prima riga. Poiché c'è una "O" nella prima riga, il potenziale orizzontale del giocatore è zero (riferimento alla riga "C" della Fig. 11.16 e consultazio-

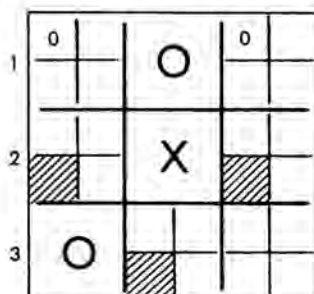


Figura 11.19: Reticolo di valutazione: potenziale della riga 1

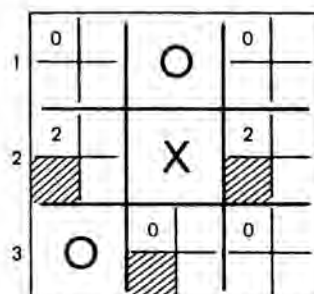


Figura 11.20: Valutazione del potenziale orizzontale

ne del valore di questa configurazione). Questo è indicato in Fig. 11.19. Osserviamo ora la riga 2; essa contiene due quadrati bianchi ed una "X". Con riferimento alla linea B della Fig. 11.16, il valore corrispondente è "due". Si entra alla locazione appropriata del reticolo, come mostrato in Fig. 11.20. Infine, si esamina la terza riga ed, essendoci una "O" in essa, il potenziale di riga è "zero", come indicato in Fig. 11.20. Il processo viene quindi ripetuto per le tre colonne. Il risultato è indicato in Fig. 11.21.

Il valore di ogni quadrato della colonna 1 è "zero", essendoci una "O" in basso. Analogamente anche per la colonna 2 il valore è "zero", mentre per la colonna 3 è "uno" per ogni quadrato, essendo tutti i tre quadrati liberi. (Riferimento alla linea A della Fig. 11.16)

Il processo viene ripetuto per le due diagonali ed i risultati sono mostrati in Fig. 11.22. Infine viene calcolato il quadrato di ogni totale. I risultati sono mostrati in Fig. 11.23. Ricordate che il totale appare nell'angolo in fondo a destra di ciascun quadrato.

0	0	O	0	1
2	0	X	2	1
O	0	0	0	1
1	2	3		

Figura 11.21: Valutazione del potenziale verticale

0	0	O	0	1
2			0	
2	0	X	2	1
O	0	0	0	1
			2	

Figura 11.22: Valutazione del potenziale diagonale

0	0	O	0	1
2	2		0	1
2	0	X	2	1
	2		3	
O	0	0	0	1
	0	2	3	

PUNTEGGIO PIU' ELEVATO

Figura 11.23: Il potenziale finale

È possibile vedere a questo punto che due quadrati (indicati con una freccia in Fig. 11.23) hanno il totale più elevato, “tre”. Questo indica dove dovremmo giocare. Un momento! Non abbiamo ancora esaminato la minaccia, cioè il potenziale del nostro avversario “O”.

Valuteremo ora la minaccia derivante da “O”, calcolando nuovamente il potenziale di ogni quadrato della matrice, ma questa volta dal punto di vista delle “O”. I valori delle posizioni per le sei combinazioni significative sono indicati in Fig. 11.24. Quando applicheremo questa strategia al nostro quadrato il punteggio più elevato è quello indicato dalla freccia. Il punteggio è “quattro” ed è quello più elevato dei due quadrati precedentemente determinati valutando il potenziale di “X”.

Impiegando il nostro algoritmo, decidiamo che si dovrebbe muovere nel quadrato 1, come indicato in Fig. 11.26.

Verifichiamo se questa è veramente la mossa appropriata assumendo che ogni giocatore esegua la miglior mossa possibile. La Fig. 11.27 riporta un possibile sviluppo del gioco. Il risultato è un pareggio.

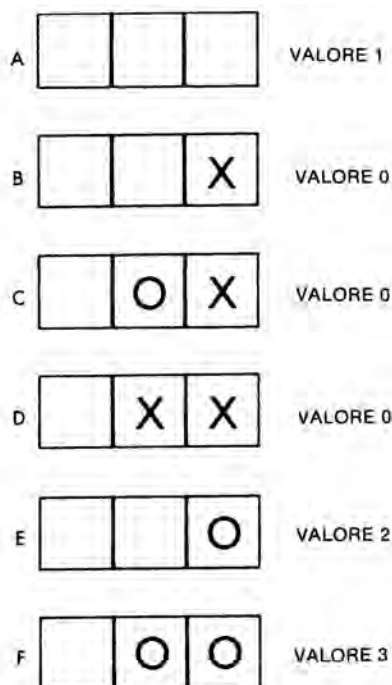


Figura 11.24: Valutazione per “O”

PUNTEGGIO PIU' ELEVATO



2	2		O	2	1
0	4			0	3
0	2		X	0	1
	2				1
O		1	0	1	1
		1	1	0	2

Figura 11.25: Valutazione del potenziale

X	O	
	X	
O		

Figura 11.26: Mossa secondo il punteggio più elevato

X	O	
	X	
O		O

X	O	
	X	
O	X	O

X	O	
	X	O
O	X	O

X	O	X
	X	O
O	X	O

(PAREGGIO)

Figura 11.27: Conclusione del gioco

Esaminiamo ora cosa sarebbe successo se non avessimo calcolato la minaccia e giocato solo secondo il potenziale più elevato di "X", come mostrato in Fig. 11.23. Questa alternativa porta a terminare il gioco come in Fig. 11.28. Anche questo gioco si risolve in un pareggio. In questo caso quindi il quadrato con il valore "quattro" non rappresenta in realtà il più alto valore strategico di quello avente valore "tre". Comunque il nostro algoritmo funziona.

Verifichiamo ora il nostro algoritmo in circostanze più difficili.

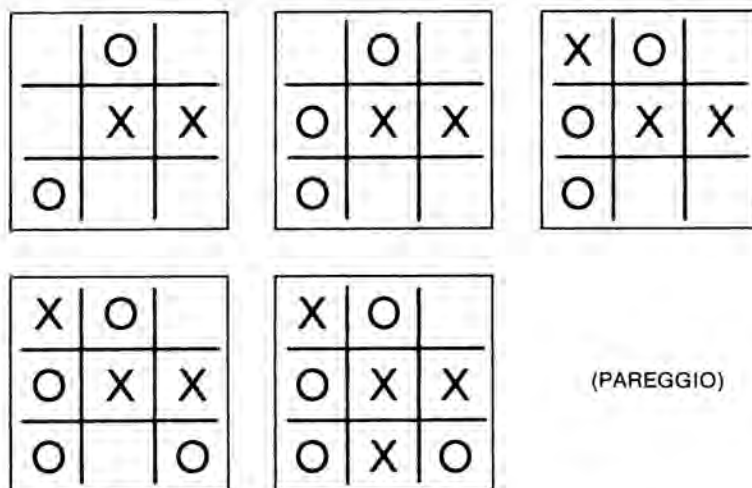


Figura 11.28: Una conclusione alternativa del gioco

### *Miglioramento dell'algoritmo*

Per verificare il nostro algoritmo, dovremo considerare situazioni particolari in cui una mossa è la migliore. Per cominciare, assumiamo che sia il turno del giocatore. La prima situazione di controllo, valutata per "X", è mostrata in Fig. 11.29 ed il potenziale di "O" è riportato in Fig. 11.30. Questa volta si presenta un problema. Il potenziale totale più elevato per "X" è "quattro" nel quadrato d'angolo in basso a destra. Comunque, se il computer eseguisse qui la sua mossa, il giocatore vincerebbe! A questo punto il nostro algoritmo deve essere affinato.

Dovremmo notare che, se esistono già due "X" in una riga la configurazione dovrebbe generare un potenziale molto alto per il terzo quadrato. Dovremmo perciò assegnargli un valore "cinque" anzichè "tre" per

0	1	0	3	O
2	3		3	
2	1	X	2	0
	3			2
2	1	X	2	0
0	3		2	4

Figura 11.29: Test #1 valutato per "X"

2	1	2	0	O
0	3		2	
0	1	X	0	1
	1			1
0	1	X	0	1
0	1		0	1

Figura 11.30: Test # 1 valutato per "O"

X		O		O	
2	2	X	2	0	
	4			2	
1	2	1	0	1	0
0	3		1	5	6

GIOCA QUI

Figura 11.31: Test #2

assicurarci di muovere automaticamente in questa posizione. Abbiamo perciò individuato ed eseguito un miglioramento per il nostro algoritmo.

La seconda situazione di prova è mostrata in Fig. 11.31. Il nostro algoritmo assegna il valore “sei” al quadrato nell'angolo in basso a destra (come indicato da una freccia in Fig. 11.31). Questa è chiaramente la mossa corretta. Essa funziona! Ora verificheremo il miglioramento che abbiamo apportato.

### *La prima mossa*

Quando la matrice è vuota, il nostro algoritmo deve decidere quale quadrato dovrebbe occupare per primo. Esaminiamo cosa fa questo algoritmo. (I risultati sono mostrati in Fig. 11.32) L'algoritmo sceglie sempre la posizione centrale. Questo è ragionevole. Comunque, si potrebbe dimostrare che questo non è indispensabile nel gioco del Tic-Tac-Toe. Infatti, può sembrare che il computer, muovendo sempre al centro, sia “noioso” o “manchi di immaginazione”. Per questo è possibile fare qualcosa, come sarà mostrato nell'implementazione finale.

1	1	1	1	1	1
1	3		2	1	3
1	1	1	1	1	1
	2	1	4		2
1	1	1	1	1	1
1	3		2	1	3

Figura 11.32: Mossa al centro

### *Un'altra prova*

Verifichiamo un'altra semplice situazione. Questa situazione è mostrata in Fig. 11.33. Anche in questo caso la mossa raccomandata è ragionevole. La situazione opposta è mostrata in Fig. 11.34, che porta certamente ad una vittoria. In questo modo il nostro algoritmo sembra funzionare. Verifichiamo una nuova trappola.



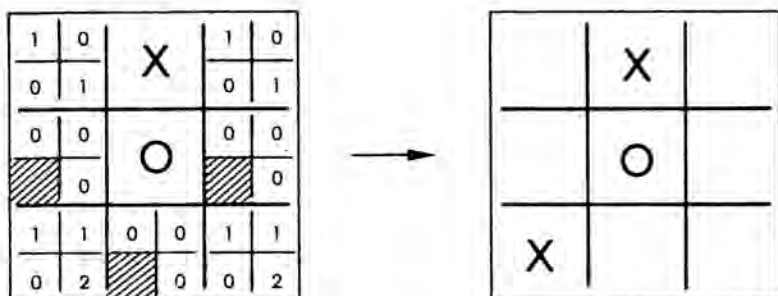


Figura 11.33: Una semplice situazione

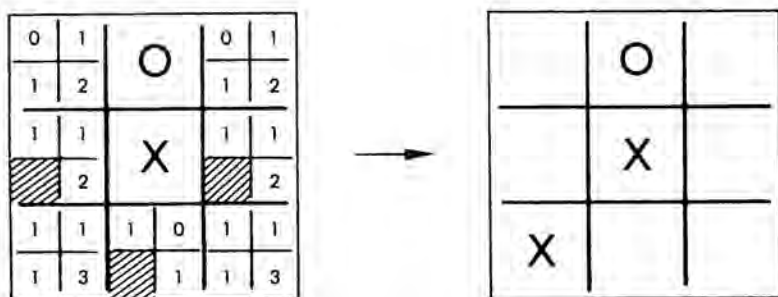


Figura 11.34: Una situazione inversa

### Una trappola

La situazione è mostrata in Fig. 11.35. È il turno delle "X", impiegando il nostro algoritmo muoveremmo in uno dei due quadrati aventi il totale "quattro". Comunque, questa volta tale mossa sarebbe un errore! Infatti facendo tale mossa, la fine del gioco è mostrata in Fig. 11.36. Si può vedere che vince "O". La mossa di "X" è una scelta sbagliata visto che c'è un modo per ottenere almeno un pareggio. La mossa corretta, infatti, dovrebbe portare ad un pareggio come mostrato in Fig. 11.37. Questa volta il nostro algoritmo non ha funzionato. Eseguiamo una semplice analisi della causa: esso muove alla posizione di un quadrato di valore "quattro" corrispondente ad un alto livello di minaccia di "O", ma si lascia un altro quadrato con un valore di minaccia uguale non protetto (vedere Fig. 11.35). Fondamentalmente questo significa che, se

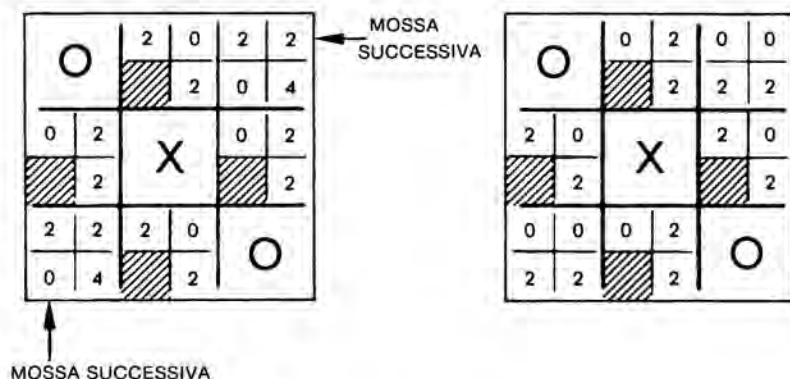


Figura 11.35: Trappola 3

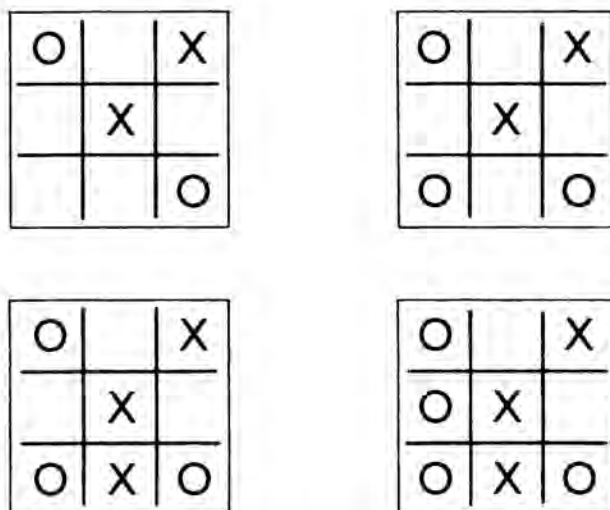


Figura 11.36: Fine del gioco

“O” è lasciato libero di muovere in un quadrato il cui potenziale di minaccia è uguale a “quattro”, esso probabilmente vincerà. In altre parole, ogni volta che la minaccia imposta da “O” raggiunge una certa soglia, l’algoritmo dovrebbe considerare strategie alternative. In questo caso, la strategia dovrebbe essere quella di posizionare una “X” in un

quadrato che è orizzontalmente o verticalmente adiacente a quello precedente in modo da creare un'imminente "minaccia di sconfitta" per "O" e quindi forzare "O" a giocare nel quadrato richiesto. In breve, questo significa che l'algoritmo dovrebbe analizzare la situazione ulteriore o migliorare, analizzare la situazione di un livello più profondo, cioè un passo avanti. Questa è chiamata un'analisi a due passi.

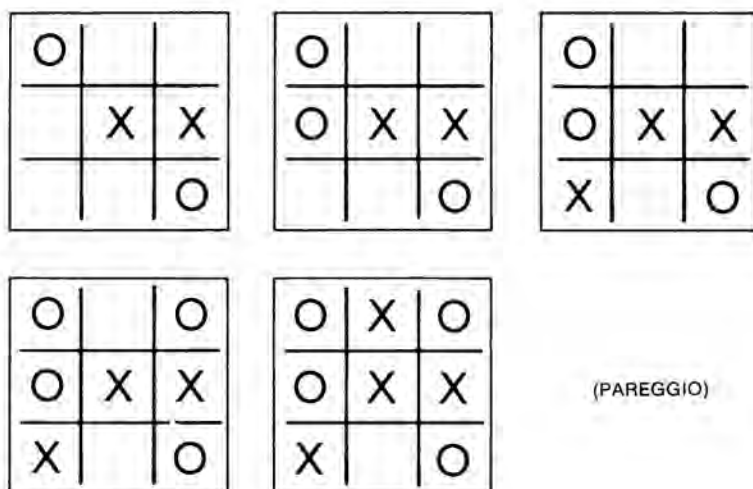


Figura 11.37: Una mossa corretta

In conclusione, il nostro algoritmo è semplice e generalmente soddisfacente. Comunque, in almeno un caso, Trappola 3 della Fig. 11.35, fallisce. Dobbiamo perciò in questo caso includere una considerazione speciale, oppure dobbiamo analizzare la situazione di un passo avanti ogni volta e guardare cosa succederebbe se posizioniamo una "X" oppure una "O" in ciascuno dei quadrati disponibili. Quest'ultima è chiaramente la soluzione più efficace. Idealmente dovremmo analizzare tutte le sequenze possibili finché non si ottiene una situazione di fine gioco. La complessità di programmazione, la memoria richiesta ed il tempo necessario per analizzare le situazioni potrebbero, comunque, rendere impraticabile questo approccio. In un gioco più complesso, come gli scacchi, sarebbe necessario impiegare una tale analisi multipla. Per esempio, impiegando una tecnica di analisi a due passi per progettare un semplice gioco degli scacchi, potrebbe non essere molto interessante o molto buona. Potrebbe essere necessario impiegare una tecnica di analisi più dettagliata a tre, quattro o più passi in modo da migliorare il gioco.

Se non è possibile spingere la valutazione ad una profondità sufficiente, l'algoritmo deve essere dotato di procedure specifiche che possono rivelare questi casi speciali. Questo è il caso della programmazione *ad hoc*, che può essere considerata non corretta concettualmente ma che, in effetti, si risolve in un programma più breve e/o in minori richieste di memoria. In altre parole, è possibile riconoscere preventivamente una situazione speciale di un gioco e quindi è possibile scrivere una routine special-purpose (per scopi speciali) che terrà conto di queste situazioni. Il programma risultante generalmente sarà più breve di uno completamente generale. Comunque, questo tipo di programma, può essere realizzato soltanto se il programmatore dispone di una comprensione iniziale eccellente del gioco.

Nel gioco del Tic-Tac-Toe il numero di combinazioni è limitato. Questo consente di esaminare tutte le possibili combinazioni che possono essere eseguite su di una matrice e determinare una procedura che tenga conto di tutti questi casi. Poiché la limitazione primaria in questo caso è la memoria disponibile, costruiamo un algoritmo *ad hoc* che realizzi questo in 1 K di memoria.

### *L'algoritmo ad hoc*

Questo algoritmo assegna un valore ad ogni quadrato della matrice in dipendenza di chi ha eseguito la mossa. Inizialmente viene assegnato un valore "zero" ad ogni quadrato sulla matrice. Ogni volta che il giocatore occupa un quadrato, il valore corrispondente di quel quadrato diventa "uno". Ogni volta che il computer occupa un quadrato, il valore di questo quadrato diventa "quattro". Questo è illustrato in Fig. 11.38. Il valore "quattro" è stato scelto in modo che sia possibile conoscere la combinazione di mosse in quella riga, semplicemente osservando il totale di ogni riga. Per esempio, se una riga è formata da una sola mossa del giocatore e da due quadrati vuoti, la sua "somma di riga" è "uno". Se il giocatore ha giocato due volte, la "somma di riga" è "due". Se il giocatore ha giocato tre volte, la somma di riga è "tre". Poiché "tre" è il totale più elevato che può essere ottenuto in una riga dove ha giocato soltanto il giocatore, il valore "quattro" è stato assegnato ad una mossa del computer. Per esempio, se il valore di una riga è "cinque", sappiamo che c'è una mossa del computer ("X"), una mossa del giocatore ("O") ed un quadrato vuoto. Le sei strutture possibili sono mostrate in Fig. 11.38. Si può osservare facilmente che i lavori di somma di riga pari a "due" oppure "otto" sono situazioni vincenti. Un valore di somma di riga pari a "cinque" rappresenta una situazione bloccata, cioè che non ha alcun valore per il giocatore. Se non è possibile una situazione vincente, allora i

STRUTTURA	VALORE DELLA SOMMA DI RIGA			
<table><tr><td></td><td></td><td></td></tr></table>				0
<table><tr><td>O</td><td></td><td></td></tr></table>	O			1
O				
<table><tr><td>O</td><td>O</td><td></td></tr></table>	O	O		2 (VITTORIA)
O	O			
<table><tr><td>X</td><td></td><td></td></tr></table>	X			4
X				
<table><tr><td>X</td><td>X</td><td></td></tr></table>	X	X		8 (VITTORIA)
X	X			
<table><tr><td>O</td><td>X</td><td></td></tr></table>	O	X		5 (BLOCCATA)
O	X			

Figura 11.38: Somme di riga

potenziali migliori sono rappresentati da un valore “uno” oppure da un valore “quattro” in dipendenza del turno di gioco.

L'algoritmo è basato su tali osservazioni. Esso preliminarmente cercherà una situazione di vittoria controllando se esiste una somma di riga con valore “otto”. In caso affermativo, verrà eseguita questa mossa. Altrimenti l'algoritmo controlla se ci si trova in una cosiddetta situazione di “trappola” nella quale due righe intersecantisi hanno ciascuna una mossa del computer nella posizione di intersezione e nient'altro (l'algoritmo viene sempre utilizzato a favore del computer). Questo è illustrato in Fig. 11.39. Esaminando la Fig. 11.39, risulta chiaro che ogni quadrato non occupato che appartiene a due righe aventi una somma di riga pari a “quattro” è una posizione di trappola dove l'algoritmo dovrebbe giocare. Questo è esattamente quello che fa.

Il diagramma di flusso completo delle analisi della matrice è mostrato in Fig. 11.40. Ora esaminiamolo più in dettaglio. Ricordate che, quando viene richiamato questo algoritmo, è sempre il turno del computer.

Da principio, esso controlla se esiste una vittoria immeditata. In pratica, esamineremo tutte le somme di riga e ne cercheremo una avente il totale di "otto". Questo corrisponde al caso in cui nella stessa riga ci sono due mosse del computer con il terzo quadrato non occupato. (Riferimento alla Fig. 11.38).

Successivamente controlleremo se esiste una possibile mossa vincente per il vincitore. Se il giocatore può vincere alla mossa successiva, l'algoritmo deve bloccare questa mossa. Per fare questo esso dovrebbe esplorare le somme di riga e vedere se ne esiste una avente un totale pari a "due", indicante una combinazione vincente per il giocatore. (Riferimento alla Fig. 11.38).

A questo punto l'algoritmo dovrebbe controllare se il computer può giocare in una qualsiasi delle posizioni di trappola precedentemente definite. (Vedere un esempio nella Fig. 11.39).

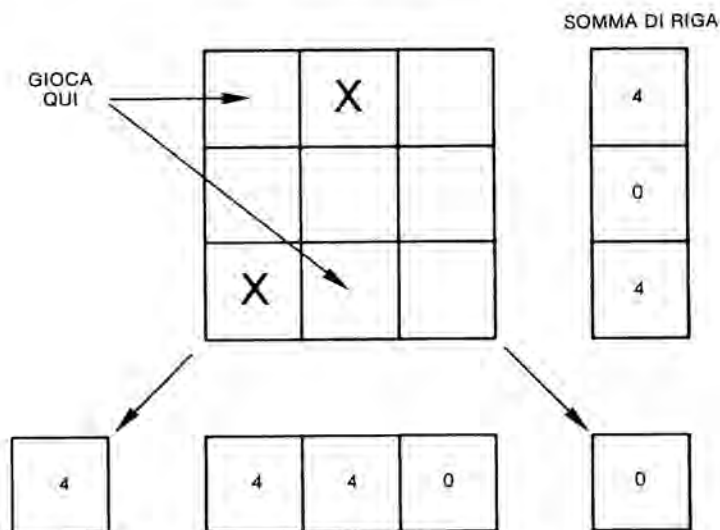


Figura 11.39: Una struttura a trappola

Un'altra caratteristica che è stata realizzata nell'algoritmo: il computer è dotato di un livello IQ variabile, cioè di un livello d'intelligenza variabile. Le mosse precedenti sono quelle che ogni "computer ragionevole" deve fare. Da questo punto in poi, comunque, l'algoritmo può lasciare che il computer esegua qualche mossa casuale ed anche possibili errori se il suo livello d'intelligenza è posto ad un valore basso. Per

fornire qualche variazione al gioco, otterremo un numero casuale, lo confronteremo con IQ e varieremo il nostro gioco in dipendenza del risultato. Se IQ è posto al massimo, il programma eseguirà sempre il ramo destro del diagramma di flusso; comunque, se IQ non è posto al massimo livello, talvolta eseguiremo il ramo di sinistra. Seguiamo il ramo di destra del diagramma di flusso. A questo punto controlliamo se si verificano due situazioni particolari che corrispondono alle mosse #1 e #4 del gioco.

Per la prima situazione, cioè per la prima mossa del gioco, l'algoritmo occuperà una posizione qualsiasi sulla scheda. In questo modo il suo comportamento sarà diverso ogni volta e, quindi, apparirà "intelligente".

Per l'altra situazione, dobbiamo osservare la mossa #4. È il turno del computer. In altre parole, il giocatore ha iniziato il gioco (mossa #1), il computer risponde (mossa #2) e quindi il giocatore esegue la sua seconda mossa (mossa #3) ed ora è il turno del computer. In breve, il giocatore ha giocato due volte ed il computer ha giocato una volta sola. A questo punto vogliamo vedere se le prime tre mosse sono state fatte lungo una delle diagonali. In caso affermativo, avendo il giocatore fatto due mosse ed il computer una, la somma di riga di una delle diagonali sarà "sei". L'algoritmo deve controllare esplicitamente questo caso. Se le prime tre mosse sono state fatte tutte lungo una diagonale, il computer deve muovere in una posizione laterale. Questa è una situazione particolare che deve essere implementata nell'algoritmo, altrimenti non si può garantire che il computer (assumendo il livello IQ più elevato) possa essere il potenziale vincitore. Questa situazione è illustrata in Fig. 11.41. Notate che, se viene impiegata una logica diretta, l'algoritmo occuperebbe una delle tre posizioni d'angolo poichè esiste una minaccia dal giocatore, che potrebbe giocare qui e portare alla situazione di trappola. I risultati di una tale azione sono mostrati in Fig. 11.42. Osservando questa illustrazione si può notare che una tale mossa porta ad una sconfitta. Esaminiamo invece cosa succede se giochiamo in uno dei lati. Questa situazione è illustrata in Fig. 11.43; il risultato è un pareggio. Chiaramente questa è la mossa da fare. È una trappola relativamente poco nota del Tic-Tac-Toe ed occorre prevederla nell'algoritmo in modo che il computer vinca.

Se non è la quarta mossa, oppure una trappola diagonale, la cosa successiva che il computer deve fare è vedere se il giocatore può impostare una trappola. (Riferimento al diagramma di flusso della Fig. 11.40). Se il giocatore può impostare una trappola, il computer gioca nel quadrato appropriato per bloccarla. Altrimenti il computer muove nel quadrato centrale, se disponibile; se non è disponibile, in una posizione casuale. Poichè questo algoritmo è stato implementato *ad hoc*, è difficile



Figura 11.40: Diagramma di flusso di analisi della matrice



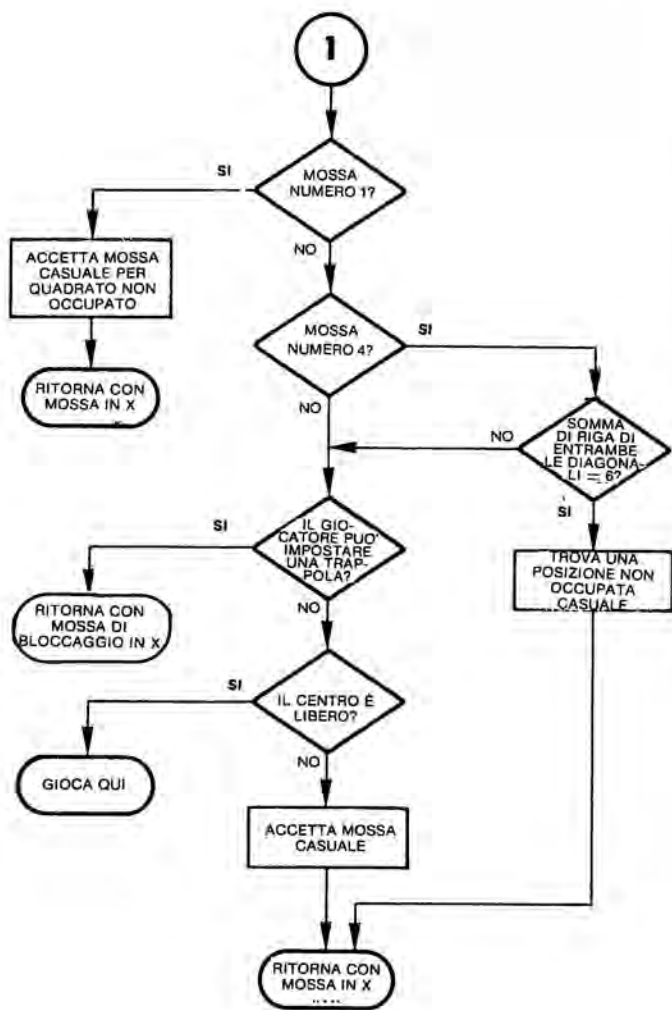


Figura 11.40: Diagramma di flusso di analisi della matrice (continua)

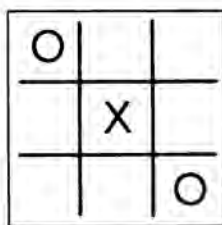
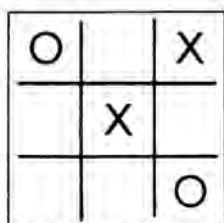
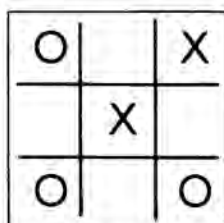


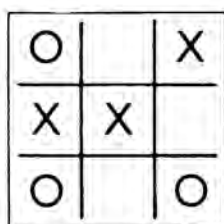
Figura 11.41: La trappola diagonale



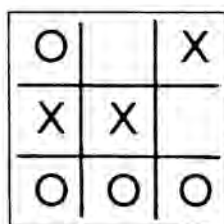
COMPUTER



GIOCATORE



COMPUTER



GIOCATORE (VINCE)

Figura 11.42: Caduta nella trappola diagonale

dimostrare che esso vince oppure ottiene un pareggio in tutti i casi. Vi suggeriamo di provare su una matrice oppure di provare il programma effettivo sulla Scheda Giochi. Scoprirete che in tutte le condizioni di verifica, il computer vince oppure realizza un pareggio. Comunque, se il computer vince ripetutamente, il suo livello IQ viene automaticamente abbassato ed, eventualmente, si consentirà al giocatore di vincere. Sono

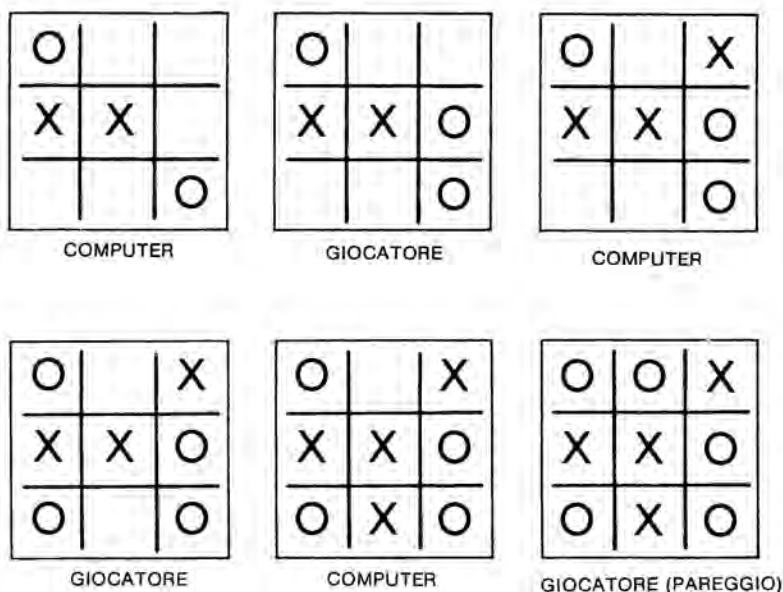


Figura 11.43: Mossa laterale

riportate in Fig. 11.44 a titolo di esempio, alcune sequenze ottenute sulla matrice effettiva.

### *Modifiche suggerite*

**Esercizio 11.1:** Fate in modo che un tasto speciale sulla Scheda Giochi, consenta, una volta premuto, la visualizzazione del livello IQ del computer.

**Esercizio 11.2:** Modificate il programma in modo che il livello IQ del computer possa essere cambiato all'inizio di ciascun gioco.

### *Ringraziamento*

L'algoritmo *ad hoc* che è stato descritto in questo paragrafo è originale. Eric Novikoff ha dato il contributo principale. Il nostro ringraziamento va anche a "Scientific American" (numeri selezionati dal 1950 al 1978) ed al Dr. Harvard Holmes che hanno fornito idee originali.

COMPUTER	GIOCATORE	COMPUTER	GIOCATORE	COMPUTER	GIOCATORE
4	5		5		6
7	1	1	6	5	4
9	8	4	7	1	9
2	(PAREGGIO)	3	2	3	7
8	5	8	9	2	(SCONFITTA)
6	3	(PAREGGIO)			6
7	9		5	5	4
1	4	3	4	8	2
(PAREGGIO)		6	9	9	1
2	5	1	2	7	(SCONFITTA)
9	1	8	7		6
7	8	(PAREGGIO)		1	5
6	3		2	4	7
(PAREGGIO)		5	1	3	2
8	5	3	7	8	9
1	7	4	6	(PAREGGIO)	
3	2	9	8	9	5
6	9	(PAREGGIO)		3	6
(PAREGGIO)			1	4	2
6	5	5	3	8	7
4	8	2	8	(PAREGGIO)	
2	3	9	6		
7	1	7	4		
(PAREGGIO)		(PAREGGIO)			

Figura 11.44: Sequenze effettive di gioco

## Strategie alternative

Si possono considerare altre strategie. In particolare, è possibile implementare un breve programma con l'impiego di tabelle di mosse che corrispondono a varie configurazioni di matrice. Le tabelle possono essere compattate tenendo conto di simmetrie, rotazioni ed in tal modo il numero di situazioni può essere ridotto. Questo tipo di approccio porta ad un programma più breve, anche se meno interessante da progettare.

**Esercizio 11.3:** *Progettate un programma per Tic-Tac-Toe che utilizzi questo tipo di tabella.*

## IL PROGRAMMA

L'organizzazione globale del programma è abbastanza semplice. Essa è mostrata in Fig. 11.42. La parte più complessa è l'algoritmo impiegato per determinare la mossa successiva del computer. Questo algoritmo, chiamato "FINDMOVE", è stato precedentemente descritto.

Esaminiamo ora l'organizzazione globale del programma. Il diagramma di flusso corrispondente è mostrato in Fig. 11.45.

1. Il livello IQ del computer viene posto al 75%.
2. Viene letto il tasto battuto dall'utente.
3. Si controlla se il tasto è una "F". Se è una "F" inizia il giocatore; altrimenti inizia il computer. In dipendenza del valore del tasto premuto, il diagramma di flusso continua nei blocchi 4 oppure 5 e quindi a 6.

Se inizia il giocatore (PLAYER è diversa da "0"), allora passiamo al lato sinistro del diagramma di flusso.

7. Il tasto, specificante la mossa, premuto dal giocatore viene letto e la mossa viene visualizzata sulla matrice.
8. Il LED corrispondente viene acceso sulla scheda. Si passa quindi al turno del computer e la variabile PLAYER nel blocco 9 è posta a "0".

All'uscita del blocco 6, nel caso del turno del computer, si passa al blocco 10.

11. A questo punto deve essere calcolata la mossa successiva eseguita dal computer.

Questo è l'algoritmo complesso che abbiamo descritto precedentemente.

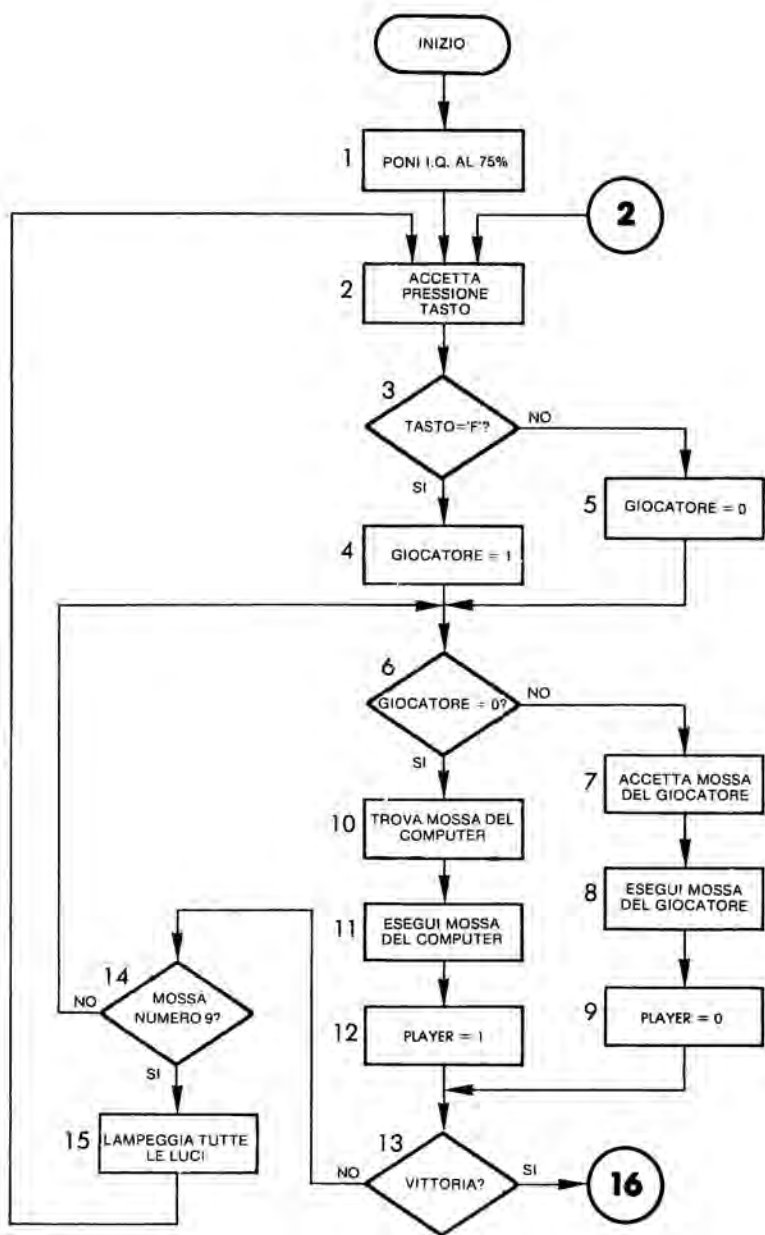


Figura 11.45: Diagramma di flusso del tic-tac-toe

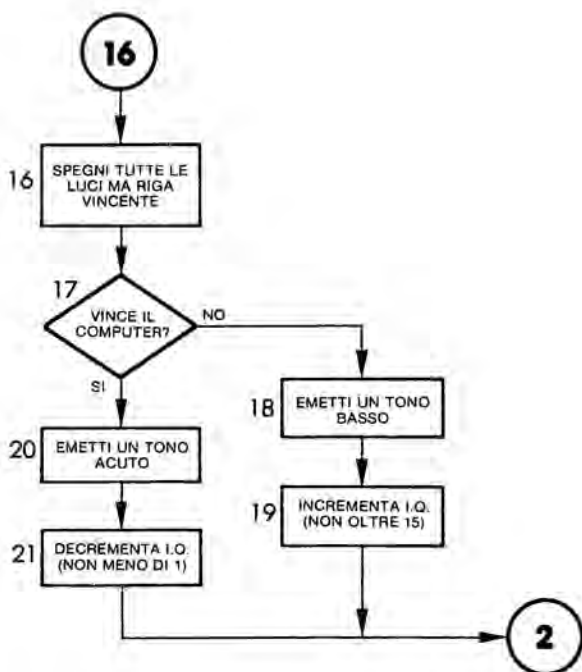


Figura 11.45: Diagramma di flusso del tic-tac-toe (continua)

11. Successivamente viene visualizzata la mossa del computer.
12. Si esegue il reset di PLAYER ad “uno” per riflettere il fatto che ora è il turno del giocatore.

Dopo che entrambi i giocatori hanno mosso, si verifica se la matrice si trova in una situazione vincente nel blocco 13. Se i LED accesi non rivelano una sequenza vincente, si passa a sinistra, sul programma di flusso.

14. Successivamente si controlla se tutte le mosse sono state esaurite: si verifica se è stata eseguita la mossa #9. Se il nono LED è acceso e non è stata rivelata una situazione vincente, è un pareggio e si procede al lampeggio di tutti i LED sulla scheda.
15. Si procede al lampeggio di tutti i LED sulla scheda. Quindi si ritorna al blocco 6 e il gioco passa all'avversario.

All'uscita del blocco 13, se c'è una situazione vincente occorre visualizzarla:

16. Vengono spenti tutti i LED eccetto i tre che visualizzano il vincitore. Successivamente, mediante l'algoritmo, si determina se ha vinto il giocatore oppure il computer.
17. Si determina se ha vinto il giocatore oppure il computer. Se ha vinto il computer si salta alla parte destra del diagramma di flusso.
18. Viene eseguito un suono basso.
19. L'IQ del computer viene decrementato (fino ad un minimo di 0).

La situazione in caso di vittoria del giocatore, mostrata nei blocchi 20 e 21, è analoga.

Il flusso generale del programma è immediato. Ora esamineremo l'informazione completa. La subroutine che analizza la situazione della matrice si chiama "ANALYZE" ed impiega la subroutine "UPDATE" per calcolare il valore delle varie posizioni della matrice.

## Strutture dei dati

La principale struttura dei dati impiegata da questo programma è una tabella lineare con tre punti d'ingresso che vengono impiegati per memorizzare gli otto possibili allineamenti di quadrati della matrice. Quando valutiamo la matrice, il programma dovrà esplorare ogni possibile allineamento di tre quadrati alla volta. Per facilitare questo processo, è stata eseguita la lista di tutti gli allineamenti possibili e la Fig. 11.46 mostra l'organizzazione di memoria.

La tabella è organizzata in tre sezioni che iniziano a RWPT1, RWPT2 ed RWPT3 (RWPT significa "row pointer", puntatore di riga). Per esempio, i primi elementi RWPT1, RWPT2 ed RWPT3 per la prima sequenza di tre quadrati, sono esaminati dalla routine di valutazione. La sequenza è: "0, 3, 6", come indicato dalle frecce in Fig. 11.43. La sequenza successiva di tre quadrati, viene ottenuta consultando il secondo ingresso in ogni tabella RWPT. Si tratta di "1, 4, 7" che è, infatti, la seconda colonna sulla nostra matrice di LED.

La tabella è stata organizzata in tre sezioni in modo da facilitarne l'accesso. Per essere in grado di accedere con successo a tutti gli elementi, sarà necessario conservare un puntatore di esecuzione che può essere impiegato come indice per un efficiente accesso alla tabella. Per esempio, se numeriamo la nostra riga generalizzata di sequenze da "0" a "7", si accederà alla "riga" 3 recuperando gli elementi agli indirizzi RWPT1 + 3, RWPT2 + 3, RWPT3 + 3. (Si tratta della sequenza "0, 1, 2" vista in Fig. 11.46).



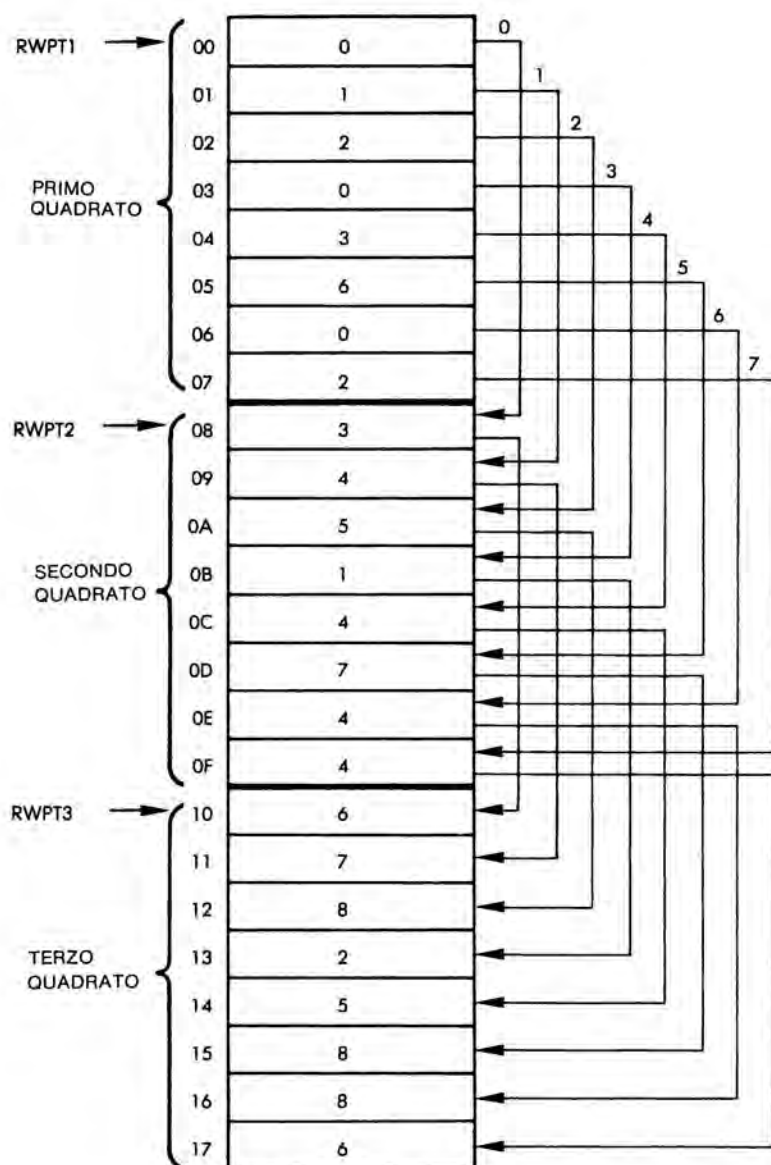


Figura 11.46: Sequenza delle righe del tic-tac-toe in memoria

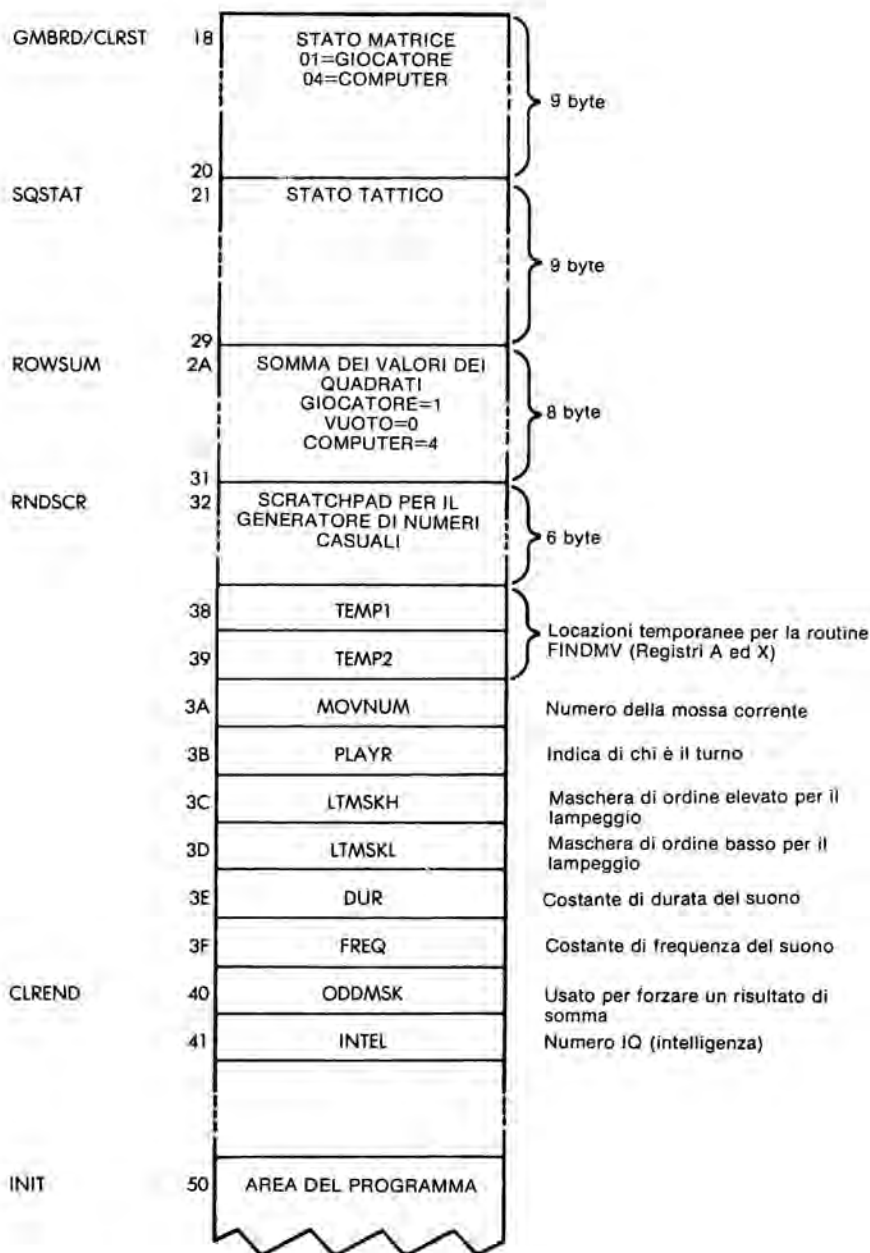


Figura 11.47; Tic-tac-toe: area bassa della memoria

## Organizzazione della memoria

La Pagina 0 contiene la tabella RWPT appena descritta, come pure le altre tabelle e variabili. Il resto dell'area bassa di memoria è mostrato in Fig. 11.47.

La tabella GMBRD occupa nove locazioni e memorizza, istante per istante, lo stato della matrice. Un valore "uno" viene impiegato per indicare una posizione occupata dal giocatore, mentre un "valore quattro" indica una posizione occupata dal computer.

La tabella SQSTAT occupa otto parole e viene impiegata per calcolare lo stato tattico della matrice.

La tabella ROWSUM occupa otto parole e viene impiegata per calcolare il valore di ciascuna delle otto righe generalizzate sul quadrato.

La tabella RNDSCR occupa sei parole e viene impiegata dal generatore di numeri casuali.

Le locazioni rimanenti sono utilizzate per le variabili temporanee, maschere e costanti, come indicato in Fig. 11.47. Il ruolo di ciascuna variabile o costante e la descrizione di ciascuna routine del programma verranno spiegate di seguito.

## Memoria alta

Le locazioni elevate della memoria sono riservate essenzialmente per i dispositivi d'ingresso/uscita. Vengono impiegate le Porte 1 e 3 e gli interrupt. La mappa di memoria corrispondente è mostrata in Fig. 11.48. Il vettore di interrupt risiede agli indirizzi A67E ed A67F. Esso verrà modificato all'inizio del programma in modo che gli interrupt saranno generati automaticamente dal timer. Questi interrupt verranno impiegati per il lampeggio dei LED sulla scheda.

## Descrizione dettagliata del programma

All'inizio di ciascun gioco, il livello di intelligenza del computer viene impostato al 75 per cento. Ogni volta che il giocatore vince, il livello IQ verrà aumentato di un punto. Ogni volta che il giocatore perde, verrà decrementato di un punto. Il livello IQ viene posto inizialmente al valore decimale 12:

START

LDA #12  
STA INTEL

Poni IQ al 75%

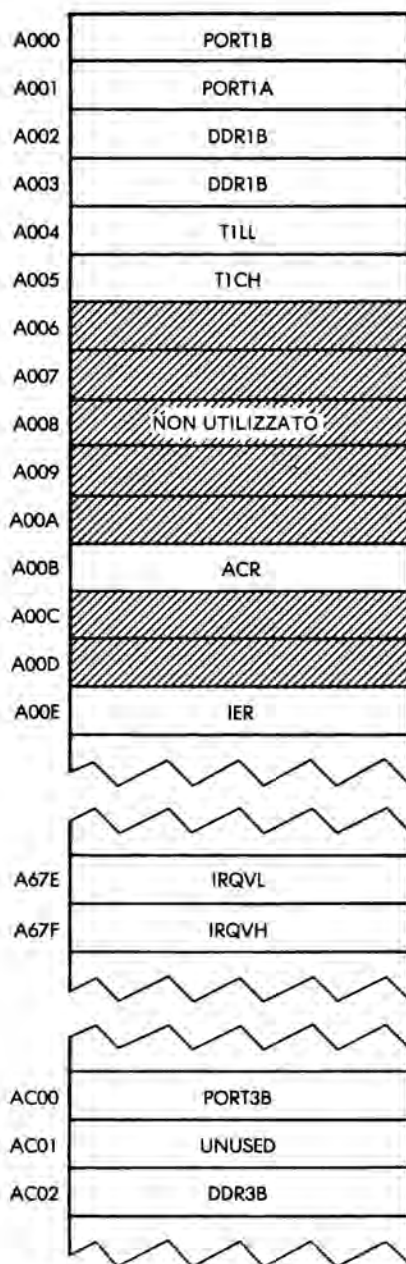


Figura 11.48: Tic-tac-toe; area alta della memoria

Quindi viene eseguita l'inizializzazione:

RESTRT      JSR INIT

Esaminiamo la subroutine INIT che è appena stata chiamata. Essa risiede all'indirizzo 0050, mentre sul listing del programma appare alla linea 0345 e successive. La prima azione della subroutine di inizializzazione è quella di azzerare tutte le locazioni basse della memoria impiegate dalle variabili del programma. Le locazioni da azzerare sono quelle tra CLRST e CLREND (vedere le linee 41 e 57 del listing del programma). Notare che in questo caso è stata utilizzata una prestazione dell'assembler raramente sfruttata - label multiple sulla stessa linea - per facilitare l'azzeramento del numero corretto di locazioni di memoria. Poiché può essere necessario introdurre più variabili temporanee nel corso dello sviluppo del programma, una label specifica è stata assegnata alla prima locazione da azzerare CLRST (locazione di memoria 18) ed un'altra all'ultima locazione da azzerare (CLREND). Per esempio, la locazione di memoria 18 corrisponde sia a CLRST che a GMBRD. L'operazione di azzeramento dovrebbe iniziare all'indirizzo CLRST e procedere verso le quaranta locazioni (CLREND-CLRST). Quindi, prima carichiamo il numero di locazioni da azzerare nel registro indice X, poi utilizziamo un ciclo per azzerare tutte le locazioni richieste:

```
INIT            LDA #0
               LDX #CLREND-CLRST
CLRALL         STA CLRST,X         Azzera locazione
               DEX
               BPL CLRALL
```

Dopo che è stata azzerata la memoria bassa, occorre attivare le due locazioni di partenza del generatore di numeri casuali. Come al solito, viene impiegato il contatore basso del timer 1:

```
               LDA TILL
               STA RNDSCR + 1
               STA RNDSCR + 4
```

Le Porte 1A, 1B e 3B vengono quindi configurate come uscite. La struttura appropriata viene quindi caricata nei registri di direzione dati:

```
               LDA #$FF
               STA DDR1A
               STA DDR1B
               STA DDR3B
```

Vengono quindi spenti tutti i LED della scheda:

```
LDA #0
STA PORT1A
STA PORT1B
```

Successivamente l'indirizzo del vettore di interrupt deve essere caricato con un nuovo puntatore. L'indirizzo da caricare in questo caso è quello del manipolatore d'interrupt, che è stato progettato per fornire il lampeggio regolare dei LED. (Questo processo è già stato spiegato ai capitoli precedenti). Il manipolatore d'interrupt risiede all'indirizzo INTVEC. Il byte elevato ed il byte basso di questo indirizzo saranno caricati nelle locazioni di memoria IRQVH ed IRQVL, rispettivamente. Un simbolo assembler speciale viene impiegato per denotare il byte basso del vettore di interrupt: #< INTVEC. Inversamente, il byte alto viene rappresentato in linguaggio assembly da #> INTVEC. Il nuovo vettore di interrupt viene caricato alle specifiche locazioni di memoria:

```
JSR ACCESS
LDA #< INTVEC
STA IRQVL      Vettore basso
LDA #> INTVEC
STA IRQVH      Vettore alto
```

Come al solito, il registro di abilitazione interrupt deve essere preliminarmente azzerato e quindi deve essere abilitato l'interrupt appropriato:

```
LDA #$7F
STA IER        Azzera registro
LDA #$C0
STA IER        Abilita interrupt
```

Il timer 1 viene impostato nel modo free-running (a corsa libera):

```
LDA #$40
STA ACR
```

Il latch del timer 1 viene caricato con il conteggio più elevato possibile, "FFFF":

```
LDA #$FF
STA TILL
STA T1CH
```

Infine vengono abilitati gli interrupt, viene azzerato il modo decimale a titolo di precauzione, e terminiamo lo stadio di inizializzazione:

```
CLI  
CLD  
RTS
```

### **Ritorno al programma principale**

Ora ci troviamo alla linea 69 del listing del programma. Leggiamo il successivo tasto premuto sulla tastiera:

```
JSR GETKEY
```

Questa è la prima mossa. Dobbiamo determinare se è una "F" oppure no. Se è una "F" il giocatore muove per primo; altrimenti muove per primo il computer. Controlliamolo:

```
LDA #01  
STA PLAYR
```

È il momento di una nuova mossa ed il contatore delle mosse viene incrementato di uno. La variabile MOVNUM viene memorizzata nella parte bassa della memoria. (Fig. 11.44). Essa viene ora incrementata:

```
PLAYLP    INC MOVNUM
```

A questo punto, PLAYR indica a chi tocca giocare. Se è posta a "zero", è il turno del computer. Se è invece posta ad "uno", è il turno del giocatore. Controlliamola:

```
LDA PLAYR  
BEQ CMPMU
```

In questo caso assumeremo che sia il turno del giocatore. Si esegue il reset di PLAYR a "zero" in modo che la mossa successiva sia riservata al computer:

```
DEC PLAYR
```

La mossa del giocatore viene ricevuta dalla subroutine PLRMV che sarà descritta in seguito. Lasciamo la mossa al giocatore:

```
JSR PLRMV
```

A questo punto, la mossa eseguita dal giocatore viene specificata dai contenuti del registro X. Poichè era una mossa del giocatore, il codice corrisponde per la rappresentazione sulla matrice dovrebbe essere "01", che verrà caricato nell'accumulatore:

LDA #01

Ora visualizzeremo la mossa sulla scheda lampeggiando il LED appropriato. Inoltre verrà automaticamente aggiornata la corrispondente ROWSUM:

JSR UPDATE

La routine UPDATE verrà descritta in dettaglio successivamente. Una volta eseguita la mossa, dovremmo controllare la possibilità di una vittoria. In caso affermativo, il giocatore ha tre LED lampeggianti in una riga ed il totale di riga corrispondente è automaticamente uguale a "tre". Perciò dobbiamo semplicemente controllare se in tutte le otto righe ce n'è una avente ROWSUM pari a tre:

LDA #03  
BNE WINTST

All'indirizzo WINTST viene eseguito un test per una configurazione vincente. Il registro Y viene caricato con "sette" ed impiegato come contatore del ciclo. Tutte le righe, dalla 7 alla 0, sono controllate se hanno un valore "tre":

WINTST      LDY #7  
TSTLP      CMP ROWSUM,4  
            BEQ WIN  
            DEY  
            BPL TSTLP

Continuiamo ora con la mossa del giocatore. Esamineremo successivamente la mossa del computer. (La mossa del computer corrisponde alle linee 83-88 del listing del programma, che non è ancora stato descritto). In questo gioco è possibile un massimo di nove mosse. Verifichiamo se abbiamo raggiunto la fine del gioco controllando il valore di MOVNUM, che contiene il numero della mossa corrente:

LDA MOVNUM  
CMP #9  
BNE PLAYLP



Questa è la fine del nostro ciclo principale. A questo punto si verifica una ramificazione alla locazione PLAYLP e si riprende l'esecuzione del programma principale.

Se a questo punto, si è raggiunta la fine del gioco si è in una situazione di pareggio, perchè non c'è ancora stato un vincitore. A questo punto tutti i LED sulla scheda dovrebbero lampeggiare e dovrebbe ri-iniziare il gioco. Imponiamo il lampeggio dei LED:

```
LDA #$FF
STA LTMSKL
STA LTMSKH
BNE DLY
```

Il ritardo viene introdotto per garantire che i LED lampeggino per un breve intervallo. Esaminiamo ora la sequenza di fine gioco.

Quando si trova una situazione vincente, può essere una vittoria del giocatore oppure del computer. Quando vince il giocatore, il totale di riga è uguale a "tre"; mentre, quando vince il computer, il totale di riga è uguale a "dodici". (Ricordate che ogni mossa del computer ha un valore "quattro" per ogni quadrato. Tre quadrati in una riga generano un totale di  $3 \times 4 = 12$ ). Se vince il computer, verrà decrementato il suo IQ:

```
WIN CMP#12
BEQ INTDN
```

A questo punto si dovrebbe verificare un salto a INTDN, dove verrà decrementato il livello di intelligenza (intelligenza abbassata).

Un suono grave verrà generato per indicare al giocatore la sua sconfitta. La costante di frequenza corrispondente è "FF" ed è memorizzata all'indirizzo FREQ:

```
INTDN    LDA #$FF
          STA FREQ
```

Il livello di intelligenza viene ora decrementato, a meno che non sia già "zero", nel cui caso rimane a quel valore:

```
LDA INTEL
BEQ GTMSK
DEC INTEL
```

Per un breve periodo di tempo la linea vincente sarà illuminata sulla scheda e verrà eseguito il suono di fine gioco. Preliminarmente azzeriamo tutti i LED sulla scheda:

```
GTMSK      LDA #0
            STA PORT1A
            STA PORT1B
```

A questo punto il numero della riga vincente è contenuto nel registro indice Y. I tre quadrati corrispondenti a quella riga saranno semplicemente recuperati dalla tabella RWPT. (Vedere Fig. 11.43). Visualizziamo il primo quadrato:

```
LDX RWPT1,Y
JSR LEDLTR
```

La routine LEDLTR verrà descritta in seguito. Essa accende il quadrato il cui numero è contenuto nel registro X. Visualizziamo il quadrato successivo:

```
LDX RWPT2,Y
JSR LEDLTR
```

Quindi il terzo quadrato:

```
LDX RWPT3,Y
JSR LEDLTR
```

A questo punto dovremmo spegnere tutti i LED lampeggianti non necessari della scheda. La nuova struttura per il lampeggio è quella della riga vincente e, quindi, dobbiamo cambiare la maschera LTMSKL:

```
LDA PORT1A
AND LTMSKL
STA LTMSKL
```

Ora eseguiamo la stessa cosa per la Porta 1B:

```
LDA PORT1B
AND LTMSKH
STA LTMSKH
```

**Esercizio 11.4:** La subroutine LEDLTR alla linea 125 del listing del programma ha appena acceso il terzo LED della riga vincente. Immediatamente dopo, iniziamo la lettura dei contenuti della Porta 1A e quindi della Porta 1B.

*Esiste la possibilità teorica che possa verificarsi un interrupt immediatamente dopo LEDLTR, che potrebbe cambiare i contenuti della Porta 1A. Questo potrebbe costituire un problema? In caso contrario, perchè non è un problema? Nel caso lo sia modificate il programma perchè operi sempre correttamente.*

A questo punto, le Porte A e B contengono la struttura appropriata per accendere la riga vincente. Se il giocatore ha vinto, le maschere di lampeggio LTMSKL ed LTMSKH contengono la stessa configurazione ed eseguiranno il lampeggio della riga. Ora siamo pronti ad eseguire il suono indicante la vittoria o la sconfitta. La durata viene posta ad "FF":

```
LDA #$FF  
STA DUR
```

La frequenza `FREQ` era stata imposta precedentemente. Dobbiamo semplicemente passare all'esecuzione:

```
LDA FREQ  
JSR TONE
```

Occorre introdurre un ritardo:

```
DLY          JSR DELAY
```

Ora siamo pronti per iniziare un nuovo gioco con un nuovo livello di intelligenza del computer:

```
JMP RESTART
```

*Ritorno a WIN*

Torniamo indietro alla linea 103 del listing del programma ed esaminiamo il caso in cui il computer non ha vinto (cioè ha vinto il giocatore). Una costante diversa di frequenza viene caricata alla locazione `FREQ`:

```
LDA #30  
STA FREQ
```

Poichè vince il giocatore, il livello di intelligenza del computer verrà aumentato, in questo caso. Comunque, prima di procedere all'incremento, occorre confrontarlo con il valore "quindici", che è il nostro massimo consentito:

```
LDA INTEL
CMP #$0F
BEQ GTMSK
INC INTEL
```

La sequenza è esattamente analoga a quella in cui il computer vince, eccetto la diversa frequenza del suono e il livello di intelligenza del computer che viene aumentato, anziché diminuito.

### *Le mosse del computer*

Ritorniamo indietro alla linea 83 del listing del programma per descrivere cosa succede quando il computer esegue una mossa. La variabile PLAYR viene incrementata e quindi viene fornito un ritardo per simulare un "tempo di riflessione" del computer:

```
COMPMV    INC PLAYR
           JSR DELAY
```

La mossa del computer è determinata dalla routine ANALYZ descritta più avanti:

```
           JSR ANALYZ
```

La mossa del computer viene caricata come un "quattro" nella locazione appropriata della scheda:

```
           LDA #04
           FSR UPDATE
```

Successivamente controlliamo su tutte le righe la possibilità di vittoria del computer, cioè per un totale di "dodici";

```
           LDA #12
WINTST     LDY #7
```

e così via. Ora ritorniamo nel programma principale precedentemente descritto.

Quando il segmento di programma precedentemente sottolineato viene confrontato con quello impiegato per la mossa del giocatore, troviamo che la differenza principale tra i due è che la mossa era specificata dalla routine ANALYZ anziché caricata dalla tastiera. Questa routine è la chiave del livello di intelligenza dell'algoritmo. Esaminiamola.

## Subroutine

### *La subroutine ANALYZ*

La subroutine ANALYZ comincia alla linea 143 del listing del programma. Il diagramma di flusso concettuale corrispondente è mostrato in Fig. 11.40. Nella subroutine ANALYZ la ODDMSK viene inizialmente posta a "zero":

```
ANALIZY    LDA #0  
           STA ODDMSK
```

Ora verifichiamo la possibilità di una vittoria del computer durante il turno successivo. Se esiste questa possibilità, chiaramente dobbiamo giocare nel quadrato vincente. Questo terminerà il gioco. Una situazione vincente è caratterizzata da un totale di "otto" nella riga corrispondente; perciò depositiamo il totale "otto" nell'accumulatore:

```
LDA #08
```

Una situazione vincente si verificherà quando i quadrati 1, 2 e 3 nelle righe hanno tutti e "tre" lo stesso valore. Impostiamo la nostra variabile filtro, X, a "tre" per il numero di righe che qualifica:

```
LDX #03
```

Ora siamo pronti per utilizzare la subroutine FINDMV:

```
JSR FINDMV
```

La routine FINDMV verrà descritta in seguito. Essa deve essere chiamata con la ROWSUM specificata in A e con il numero di volte che è stata trovata un'uguaglianza in X. Essa controllerà sistematicamente tutte le righe e quadrati. Se si trova un quadrato, si esce con il numero del quadrato specificato in X ed il flag Z posto a "0". Verifichiamolo:

```
BNE DONE
```

Se è stata trovata una mossa vincente, si esce dalla routine ANALYZ. Sfortunatamente questo non è il caso più comune ed occorre eseguire altre analisi.

La particolare situazione successiva da controllare è quella di vedere se il giocatore ha una mossa vincente. In caso affermativo, occorre

bloccarla. Una situazione vincente per il giocatore è indicata da un totale di riga pari a "due". Carichiamo "2" nell'accumulatore e ripetiamo il processo precedente:

```
LDA #02  
LDA #03  
JSR FINDMV  
BNE DONE
```

Se il giocatore potesse eseguire una mossa vincente, è questo il quadrato in cui il computer deve giocare ed usciamo a DONE; altrimenti la situazione dovrebbe essere ulteriormente analizzata.

Ora controlleremo se il computer può implementare una trappola. Una trappola corrisponde ad una situazione in cui una mossa del computer è già stata fatta in qualche riga. Dovremmo cercare di giocare nell'intersezione di due righe contenenti mosse del computer. Questo è stato spiegato precedentemente nel corso della descrizione dell'algoritmo. Questa situazione è caratterizzata da  $A = 4$  ed  $X = 2$ . Carichiamo i registri con i valori appropriati e chiamiamo la routine FINDMV:

```
LDA #04  
LDX #02  
JSR FINDMV  
BNE DONE
```

Se il test è soddisfatto si esce a DONE; altrimenti si procede in basso nel diagramma di flusso di Fig. 11.40.

È a questo punto che il computer può dimostrare la sua intelligenza oppure un gioco poco astuto. Il comportamento del computer sarà determinato dal suo livello di intelligenza. Ora otterremo un numero casuale e lo confronteremo all'IQ del computer. Se il numero casuale supera l'IQ del computer, procederemo per la parte sinistra del diagramma di flusso della Fig. 11.40 e faremo una mossa poco astuta (cioè una mossa casuale). Se il numero casuale non supera l'IQ del computer, eseguiremo una mossa intelligente passando per la parte destra del diagramma di flusso. Generiamo il numero casuale:

```
JSR RANDOM
```

Tronchiamo il numero casuale al suo byte di destra in modo che esso non superi quindici:

```
AND #$0F
```

e lo confrontiamo con l'IQ corrente del computer:

```
CMP INTEL  
BEQ OK  
BCS RNDMV
```

Se il numero casuale è maggiore del livello IQ memorizzato in INTEL, si salta a RANDMV e si esegue una mossa casuale. A questo punto, assumiamo che il numero casuale non sia maggiore del livello IQ e che il computer esegua una mossa intelligente. Ora procediamo dalla linea 162 (locazione "OK").

Dapprima controlleremo se questa è la mossa #1; quindi se è la mossa #4. Controlliamo se è la mossa #1:

```
OK          LPX MOVNUM  
            CPX #1
```

Se è la mossa #1, occupiamo un quadrato qualsiasi:

```
BEQ RNDMV
```

Controlliamo se è la mossa #4:

```
CPX #4
```

Se non è la mossa #4, controlliamo se il giocatore può impostare una trappola. Questo verrà eseguito alla locazione TRAPCK. In questo caso assumiamo che sia la mossa #4.

```
BNE TRAPCK
```

Questa sezione controllerà entrambe le diagonali per la possibilità della sequenza giocatore-computer-giocatore. Se si trova questa sequenza, si giocherà lateralmente. Altrimenti si ritorna indietro al corpo di questa routine e si controlla se il giocatore può impostare una trappola. La combinazione giocatore-computer-giocatore in una riga viene rivelata dal totale "sei" della riga. Quindi carichiamo il valore "sei" nell'accumulatore e controlliamo la diagonale corrispondente. Per coincidenza, le diagnosi corrispondono al sesto e settimo ingresso della nostra tabella RWPT. (Vedere la Fig. 11.46). Eseguiamolo:

```
LDX #6  
TXA
```

CMP ROWSUM,X  
BEQ ODDRND

Se si trova un'uguaglianza si salta all'indirizzo ODDRND, dove si giocherà una posizione laterale. Questo verrà descritto in seguito. Se, non si trova un'uguaglianza, controlliamo la diagonale successiva:

INX  
CMP ROWSUM,X  
BEQ ODDRND

Se, a questo punto, anche questo test non è soddisfatto, per la seconda diagonale, controlleremo se il giocatore può impostare una trappola.

*Controllo per vedere se il giocatore può impostare una trappola sparire normalmente (TRAPCK)*

Si identifica la possibilità di una trappola per il giocatore (analogamente nel caso del computer) quando due righe che si intersecano contengono una sola mossa del giocatore. Questo è stato spiegato nella descrizione del precedente algoritmo. Il valore di una riga che è dedicata ad una trappola è perciò uguale ad "uno" (una mossa del giocatore). Occorre quindi impostare i parametri in questo modo: A = 1 ed X = 2 prima di richiamare la routine FINDMV:

TRAPCK     LDA #1  
            LDX #2  
            JSR FINDMV  
            BNE DONE

Se si trova la locazione appropriata per una trappola, questa è la mossa successiva da giocare. Altrimenti, se possibile, il computer muove al centro, oppure, se il centro è occupato, esegue una mossa casuale laterale.

LDX GMBRD + 4  
BNE RNDMU  
LDX #5  
BNE DONE

*Esecuzione di una mossa casuale laterale*

I quattro lati della matrice sono numerati esternamente 2, 4, 6 ed 8, oppure internamente, 1, 3, 5 e 7. Qualsiasi numero interno dispari indica



una posizione laterale. Se si vuole occupare una posizione laterale, carichiamo semplicemente il valore "uno" in ODDMSK, garantendo in tal modo che il numero casuale generato sarà uno di quelli ai quattro angoli. Questo viene eseguito caricando all'indirizzo ODDRND:

```
ODDRND    LDA #1  
           STA ODDMSK
```

Comunque, in generale, possiamo voler eseguire una mossa casuale. Questo verrà eseguito generando ed impiegando un qualsiasi numero casuale che è ragionevole, cioè ponendo ODDMSK a "0" prima di entrare all'indirizzo RNDMV. Otteniamo un numero casuale:

```
RNDMV     JSR RANDOM
```

Eliminiamo il byte di sinistra:

```
AND #$0F
```

Quindi eseguiamo l'OR di questo numero casuale con la struttura memorizzata in ODDMSK. Se la maschera è stata posta a "0", non si avrà alcun effetto sul numero casuale. Se invece la maschera era stata posta ad "1", potrebbe risultare la mossa in uno degli angoli (in questo caso il centro è occupato):

```
ORA ODDMSK
```

Poichè il numero casuale che è stato generato è compreso tra "0" e "15", dobbiamo assicurarci che non superi "9"; nel qual caso non può essere utilizzato:

```
CPM #9  
BCS RNDMV
```

Dobbiamo ora controllare che lo spazio in cui vogliamo muoverci non sia occupato. Carichiamo il numero del quadrato nel registro indice X e verifichiamo lo stato del quadrato leggendo l'ingresso appropriato della tabella GMBRD. (Vedere la mappa di memoria in Fig. 11.47):

```
TAX  
LDA GMBRD,X
```

Se in questo quadrato c'è un qualsiasi ingresso diverso da "0", significa

che è occupato e dobbiamo generare un altro numero casuale:

### BNE RNDMV

Abbiamo selezionato un quadrato valido ed eseguiamo in esso la mossa. Quando usciamo da questa routine, il numero esterno del LED dovrebbe essere contenuto in X. Questo si ottiene sommando "1" ai contenuti correnti di X, che rappresentano il numero interno del LED:

	INX
DONE	RTS

### *Subroutine FINDMV*

Questa subroutine valuterà la matrice fino a trovare un quadrato che soddisfa le specifiche nei registri A ed X. L'accumulatore A contiene una somma di riga specifica, che un riga deve avere per essere scelta. Il registro indice X specifica il numero di volte che un particolare quadrato deve appartenere ad una riga la cui somma di riga è uguale a quella specificata da A.

La subroutine FINDMV inizia con lo stato del quadrato pari a "0" per ogni quadrato della matrice. Ogni volta che trova un quadrato che soddisfa la specifica sulla somma di riga, essa aumenterà il suo stato di "1". Quindi, alla fine del processo di valutazione, un quadrato avente lo stato pari ad "1", è un quadrato che ha soddisfatto una volta le specifiche relative alla somma di riga. Un quadrato avente stato "2", ha soddisfatto due volte la specifica e così via.

La scelta finale viene eseguita da FINDMV, che controlla in successione ogni quadrato. Non appena essa trova un quadrato il cui stato è uguale al numero contenuto nel registro X, seleziona quel quadrato come quello che soddisfa la specifica iniziale.

Il diagramma di flusso completo di FINDMV è mostrato in Fig. 11.49. Essenzialmente la subroutine opera in tre passi. Questi sono riportati in Fig. 11.49. Il Passo 1 è la fase di inizializzazione. Il Passo 2 corrisponde alla scelta di tutti i quadrati che soddisfano le specifiche sulla somma di riga contenute nel registro A. Lo stato di ogni quadrato vuoto di una riga che soddisfa questa specifica viene aumentato di uno quando tutte le righe sono state esplorate. Il Passo 3 è la fase di scelta finale. In questa fase ogni quadrato viene osservato a turno finché non se ne trova uno il cui stato è uguale al valore contenuto in X. Appena viene identificato tale quadrato, il processo si arresta. Questo è il quadrato che verrà giocato dal computer. Se non si trova un tale quadrato, si esce dalla

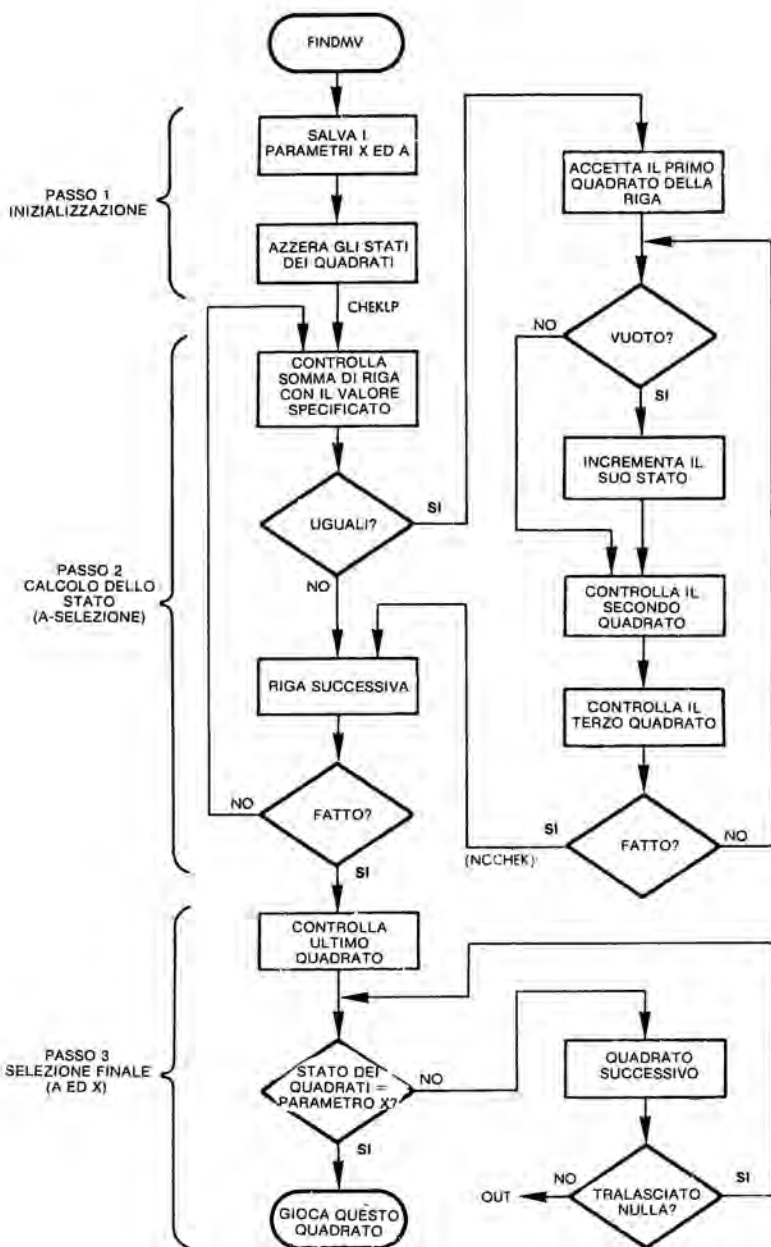


Figura 11.49: Diagramma di flusso di FINDMV

routine con il registro indice X decrementato a "0" e questo verrà utilizzato come flag di guasto o di errore per la routine chiamante.

Esaminiamo ora il programma corrispondente. Esso inizia alla linea 204 del listing del programma.

### *Passo 1: Inizializzazione*

I registri X ed A saranno impiegati nel corpo di questa subroutine. I loro contenuti iniziali devono essere preservati in locazioni di memoria temporanee. Gli indirizzi TEMP1 e TEMP2 sono impiegati per questo scopo. (Vedere Fig. 11.47 per la mappa di memoria).

Preserviamo X ed A:

```
FINDMV    STX TEMP2  
          STA TEMP1
```

Viene quindi azzerato lo stato della matrice. Lo stato di ogni quadrato deve essere posto a "0". Questo viene eseguito caricando il valore "0" nell'accumulatore e quindi eseguendo 9 volte un ciclo che azzerava lo stato di ogni quadrato in successione:

```
          LDA #0  
          LDY #8  
CLRLP     STA SQSTAT,4  
          DEY  
          BPL CLRLP
```

### *Passo 2: Calcolo dello stato di ogni quadrato*

Verranno ora esaminate in successione tutte le otto possibili somme di riga. Se la somma di riga è uguale al valore specifico nell'accumulatore, lo stato di ogni quadrato vuoto all'interno della riga specifica verrà incrementato di "1". Se la somma di riga corrente non soddisfa il test si passa alla successiva. Il registro indice Y viene impiegato come puntatore di riga. La tabella RWPT, descritta all'inizio di questo programma e mostrata in Fig. 11.46, sarà impiegata per recuperare in modo efficiente i tre quadrati da ciascuna riga. Dapprima inizializziamo il contatore:

```
LDY #7
```

Ora controlleremo il valore della corrispondente somma di riga:

```
CHEKLP    LDA TEMP1
           CMP ROWSUM,Y
           BNE NOCHEK
```

Assumiamo a questo punto che la somma di riga sia quella corretta. Dobbiamo ora esaminare ciascuno dei tre quadrati della riga. Se un quadrato è vuoto incrementeremo il suo stato. Il primo passo è di ottenere il valore del quadrato consultandolo nella tabella, impiegando il registro indice Y come spostamento ed impiegando gli indirizzi RWPT1, RWPT2 ed RWPT3 in successione come punti d'ingresso nella tabella delle righe. Eseguiamolo per il primo quadrato:

```
LDX RWPT1,Y
```

Il registro indice X ora contiene il numero del quadrato. Se il quadrato è vuoto, una nuova subroutine, CNTSUB, viene impiegata per incrementare il suo stato:

```
JSR CNTSUB
```

Essa verrà descritta in seguito.

Ora eseguiamo la stessa cosa per il secondo e terzo quadrato:

```
LDX RWPT2,Y
JSR CNTSUB
LDX RWPT3,Y
JSR CNTSUB
```

Ora abbiamo esplorato completamente una riga. Vediamo se occorre controllare altre righe:

```
NOCHEK    DEY
           BPL CHECKLP
```

Il processo viene ripetuto fino ad aver controllato tutte le righe. A questo punto, entriamo al passo 3 di FINDMV. (Riferimento al diagramma di flusso di Fig. 11.49).

### *Passo 3: Scelta finale*

Il registro indice X verrà impiegato come puntatore del quadrato. Esso inizia con il quadrato #9 e continua ad esaminare quadrati finchè

non ne trova uno che soddisfa le specifiche aggiuntive del registro X, cioè il numero di volte che un dato quadrato appartiene ad una riga con l'appropriato valore della somma di riga. Inizializziamo X:

LDX #9

Ora confrontiamo il valore dello stato del quadrato con il valore del parametro specificato:

```
FNMTCH    LDA TEMP2
           AND SQSTAT-1,X
```

Se lo stato del quadrato è uguale al valore del parametro, selezioniamo questo quadrato:

BNE FOUND

Altrimenti passiamo a provare il successivo:

```
           DEX
           BNE FNMTCH
FOUND      RTS
```

*Esercizio 11.5: Perché per verificare un'eguaglianza di quadrati, in precedenza, abbiamo utilizzato "AND" e "BNE" anziché "CMP" e "BEQ"? (Suggerimento: decidere quale differenza si dovrebbe avere nella strategia del programma).*

### Subroutine COUNTSUB

Questa subroutine viene impiegata esclusivamente dalla subroutine FINDMV ed incrementa lo stato del quadrato il cui numero è contenuto nel registro X, se il quadrato è vuoto. Dapprima essa esamina lo stato del quadrato consultando il suo codice nella tabella GMBRD:

```
CNTSUB    LDA GMBRD,X
           BNE NOCNT
```

Se il quadrato è occupato, si esce. In caso contrario, il valore dello stato del quadrato viene incrementato:

```
NOCNT     INC SQSTAT,X
           RTS
```

### *Subroutine UPDATE*

Ogni volta che viene eseguita una mossa, occorre visualizzarla sulla scheda. Conseguentemente, occorre memorizzare il codice appropriato nella rappresentazione della matrice, cioè nella tabella GMBRD. Infine, occorre calcolare e memorizzare le locazioni appropriate. Queste funzioni vengono eseguite dalla subroutine UPDATE.

Il codice del giocatore è contenuto nell'accumulatore. La posizione in cui viene eseguita la mossa è contenuta nel registro X. Poichè nel registro X è contenuto il valore di un LED esterno, esso viene prima decrementato in modo da ottenere il numero effettivo del LED interno:

UPDATE      DEX

Il valore deve ora essere memorizzato nella locazione appropriata della tabella GMBRD, che contiene la rappresentazione interna della matrice:

STA GMBRD,X

Notate che il valore di X viene impiegato semplicemente come spostamento nella tabella. Comunque, l'accumulatore contiene il codice appropriato che viene semplicemente scritto alla locazione specifica. A questo punto, UPDATE dovrebbe visualizzare la mossa sui LED. Occorre però decidere prima se l'accensione del LED è stazionaria o lampeggiante. Per fare questo la subroutine deve determinare se è la mossa del giocatore oppure del computer. Questo viene eseguito esaminando il codice contenuto nell'accumulatore. Se il codice è "quattro" è la mossa del computer. Invece se il codice è "uno" è la mossa del giocatore. Esaminiamolo:

CMP #04  
BEQ NOBLNK

Se è la mossa del computer, si verifica su salto all'indirizzo NOBLNK; altrimenti si procede. Assumiamo che sia la mossa del giocatore:

JSR LIGHT

La subroutine LIGHT viene impiegata per il lampeggio di bit e verrà descritta in seguito. All'uscita da LIGHT l'accumulatore contiene il bit nella posizione richiesta per il lampeggio del LED. A questo punto deve essere aggiornata la maschera di lampeggio:

ORA LTMSKL  
STA LTMSKL

Se il carry era "zero" all'uscita da LIGHT uno dei bit da zero a sette è stato posto ad uno:

BCC NOBLNK

Altrimenti, se il carry era stato posto ad 1, potrebbe significare che il LED #9 deve essere acceso, cioè che la parte di ordine elevato della maschera deve essere modificata. Facciamolo:

LDA #01  
STA LTMSKH

A questo punto, la maschera dei LED è configurata correttamente e possiamo fornire l'ordine di accensione dei LED:

NOBLNK JSR LEDLTR

La routine LEDLTR accende i LED specificati dal registro X. Notate che se fosse stata una mossa del computer, questo LED sarebbe rimasto acceso in modo stazionario. Per una mossa del giocatore, invece, il LED sarebbe stato acceso e spento automaticamente al verificarsi degli interrupt.

Successivamente, dobbiamo aggiornare tutte le somme di riga. Il registro indice X viene impiegato come puntatore di riga. Verranno osservate tutte le otto righe in successione. Il bit carry viene azzerato, anticipando l'addizione:

LDX #7  
ADDROW CLC

Innanzitutto viene esaminato il primo quadro della riga otto:

LDY RWPT1,X

Notate che il registro indice Y conterrà il numero interno del quadrato successivo a questa istruzione. Questo verrà immediatamente sfruttato per un'altra operazione indicizzata. Verranno letti i contenuti del quadrato in modo che possa essere calcolata la nuova somma di riga. (La



somma di riga per la riga corrente può variare oppure no. Nessuna previsione speciale è stata fatta per restringere la ricerca alle due o tre righe influenzate). Tutte le righe vengono esaminate in successione e tutte le somme di riga vengono ricalcolate in modo da semplificare il programma.

Otteniamo il valore del quadrato corrente:

LDA GMBRD,Y

Si accede alla tabella GMBRD con l'impiego del registro indice Y come valore dello spostamento. Notate che le due istruzioni mostrate precedentemente implementano un'indicizzazione a due livelli. Questa è la tecnica più efficiente per il recupero dei dati. A questo punto l'accumulatore contiene il valore del primo quadrato. Esso verrà sommato al valore dei due quadrati successivi. Quindi il processo verrà ripetuto:

LDY RWPT2,X  
ADC GMBRD,Y

Il numero del secondo quadrato è stato prelevato dall'istruzione LDY ed il suo valore memorizzato in Y. L'istruzione di addizione preleva il valore effettivo di quel quadrato da GMBRD e somma tale valore all'accumulatore. Questo processo viene eseguito un'altra volta per il terzo quadrato:

LDY RWPT3,X  
ADC GMBRD,Y

Il valore finale contenuto nell'accumulatore viene quindi memorizzato nella tabella ROWSUM nella posizione specificata dal valore del registro indice X (l'indice della riga):

STA ROWSUM,X

Verrà quindi esplorata la riga successiva:

DEX  
BPL ADDROW

Se X diventa negativo, si esce:

RTS

### *Subroutine LED LIGHTER*

Questa subroutine assume che, all'atto della chiamata, il registro X contenga il numero interno del LED che deve essere acceso sulla scheda. Quindi la subroutine accenderà quel LED usando la subroutine LIGHT, che converte un numero nel registro X in una struttura di bit nell'accumulatore per l'accensione del LED specifico:

```
LEDLTR    JSR LIGHT
```

A questo punto, occorre aggiornare la Porta A, oppure la Porta B. Assumiamo inizialmente che sia la Porta 1A (se non è la Porta 1A possiamo trovarlo esaminando il bit carry successivo, quindi la configurazione contenuta nell'accumulatore è formata da tutti zeri e non varierà il valore della Porta 1A):

```
ORA PORT1A  
STA PORT1A  
BCC LTRDN
```

Viene verificato il bit carry. Se esso è stato posto ad 1 dalla subroutine LIGHT, allora occorre accendere il LED #9. Questo viene eseguito inviando un "1" alla Porta 1B:

```
LDA #1  
STA PORTB  
RTS
```

### *Subroutine PLRMV mossa del giocatore)*

Questa subroutine ottiene una mossa corretta dal giocatore. Essa attira l'attenzione del giocatore ed attende un ingresso da tastiera. Se viene premuto un tasto diverso da quelli da 1 a 9, verrà ignorato. Ogni volta che la subroutine accetta una mossa, verifica che il quadrato relativo sulla scheda sia effettivamente vuoto. Se il quadrato non è vuoto la subroutine ignora la mossa del giocatore. Dapprima generiamo un chirp (cinguettio) per attirare l'attenzione del giocatore:

```
PLRMV     LDA #$80  
          STA DUR
```

LDA #10  
JSR TONE

Acquisiamo ora la chiusura del tasto:

KEYIN JSR GETKEY

Dobbiamo ora controllare se il tasto premuto è compreso tra 1 e 9.  
Vediamo innanzitutto se non è maggiore o uguale a 10:

CMP #10  
BCS KEYIN

Verifichiamo ora che non sia uguale a "zero":

TAX  
BEQ KEYIN

Infine verifichiamo che non corrisponda ad un quadrato già occupato:

LDA GMBRD-1,X  
BNE KEYIN  
RTS

*Esercizio 11.6: Modificate la precedente subroutine PLRMV in modo che venga generato un nuovo chip ogni volta che il giocatore esegue una mossa non corretta. Per informare il giocatore che ha eseguito una mossa non corretta, dovrete generare una sequenza di due chirp, impiegando un suono diverso da quello precedente.*

### *Subroutine LIGHT*

Questa subroutine accetta un numero di LED nel registro X. Essa ritorna con la configurazione da inviare ai LED contenuta nell'accumulatore. Se deve essere acceso il LED #9 (X=8), il bit carry sarà uguale ad uno. Questa subroutine è immediata ed è già stata descritta precedentemente:

LIGHT STX TEMP1  
SEC  
ROL A

```

DEX
BPL SHIFT
LDX TEMP1
RTS

```

### *Subroutine DELAY*

Questa è una classica subroutine di ritardo. Essa impiega due cicli annidati con alcune istruzioni extra all'interno del ciclo predisposte per perdere tempo:

```

DELAY      LDY #$FF
DL1        LDX #$FF
DL2        ROL DUR
           ROR DUR
           DEX
           BNE DL2
           DEY
           BNE DL1
           RTS

```

### *Routine di manipolazione interrupt*

Ogni volta che viene ricevuto un interrupt, verranno complementati i LED appropriati (spenti se accesi e viceversa). Le posizioni del LED da far lampeggiare sono specificate dai contenuti delle maschere LTMSK. Due byte sono impiegati nella memoria per le metà bassa ed alta, rispettivamente. (Vedere la Fig. 11.47 per la mappa di memoria).

L'operazione di commutazione dei bit viene eseguita mediante un'istruzione di OR esclusivo che è equivalente ad una complementazione logica. Poichè questa routine impiega l'accumulatore, i contenuti di A devono essere preservati all'inizio della routine. La subroutine è la seguente:

```

INTVEC     PHA
           LDA PORT1A
           EOR LTMSKL
           STA PORT1A
           LDA PORT1B
           EOR LTMSKH
           STA PORT1B

```

LDA TILL  
PLA  
RTI

**Esercizio 11.7:** *Notate la precedente istruzione LDA TILL. L'istruzione successiva di questa subroutine è PLA. Essa riscriverà i contenuti dell'accumulatore con le parole prelevate dallo stack. I contenuti dell'accumulatore, come letti da TILL, saranno perciò immediatamente distrutti. Questo è un errore di programmazione accidentalmente lasciato nel programma? In caso contrario qual'è il suo scopo? (Suggerimento: questa situazione è stata incontrata precedentemente. Fate riferimento ad uno dei capitoli precedenti.)*

*Subroutine INITIALIZE*

Questa subroutine è stata descritta precedentemente nel programma principale.

*Subroutine RANDOM e TONE*

Queste due subroutine sono state descritte nei programmi precedenti.

## SOMMARIO

Questo programma è il più complesso di quelli sviluppati. Sono stati presentati diversi algoritmi, un'implementazione completa di un algoritmo *ad hoc* è stata studiata nei minimi dettagli. I lettori interessati ai giochi di strategia e di programmazione sono incoraggiati ad implementare un algoritmo alternativo.

LINE	#LOC	CODE	LINE
0002	0000		'TICTAC'
0003	0000		PROGRAMMA PER ESEGUIRE IL TIC-TAC-TOE SUL SYM-1
0004	0000		COMPUTER CON UNA MATRICE DI LED 3 x 3 ED UNA
0005	0000		TASTIERA ESADEC ALL'INIZIO DEL GIOCO, SE VIENE
0006	0000		PREMUTO IL TASTO "F", INIZIA IL GIOCATORE, CON
0007	0000		QUALSIASI ALTRO TASTO, INIZIA IL COMPUTER.
			SUCCESSIVAMENTE.
0008	0000		PER ESEGUIRE UNA MOSSA, PREMETE IL TASTO
0009	0000		CORRISPONDENTE AL NUMERO DEL QUADRATO
			DESIDERATO
0010	0000		
0011	0000		LINKAGES:
0012	0000		
0013	0000	GETKEY	= \$ 100
0014	0000	ACCESS	= \$ 8B86
0015	0000		
0016	0000	I/O:	
0017	0000		
0018	0000	PORT1A	= \$ A001 ; ** 6522 VIA #1...
0019	0000	DDR1A	= \$ A003
0020	0000	PORT1B	= \$ A000
0021	0000	DDR1B	= \$ A002
0022	0000	IER	= \$ A00E ; REGISTRO ABILITAZIONE
			INTERRUPT.
0023	0000	ACR	= \$ A00B ; REGISTRO DI CONTROLLO
			AUSILIARIO.
0024	0000	T1LL	= \$ A004 ; LATCH BASSO SUL TIMER 1.
0025	0000	T1CH	= \$ A005 ; LATCH ALTO SUL TIMER 1.
0026	0000	PORT3B	= \$ AC00 ; ** 6522 VIA #3....
0027	0000	DDR3B	= \$ AC02
0028	0000	IRQVL	= \$ A67E
0029	0000	IRQVH	= \$ A67F
0030	0000		
0031	0000		TABELLA DEI QUADRATI NELLA MATRICE.
0032	0000		
0033	0000		* = 0
0034	0000		
0035	0000	00	
0035	0001	01	
0035	0002	02	
0035	0003	00	
0035	0004	03	
0035	0005	06	
0035	0006	00	
0035	0007	02	
0036	0008	03	
0036	0009	04	
0036	000A	05	
0036	000B	01	
0036	000C	04	
0036	000D	07	
0036	000E	04	
0036	000F	04	
0037	0010	06	
0037	0011	07	
0037	0012	08	
0037	0013	02	
0037	0014	05	
0037	0015	08	
0037	0017	06	

Figura 11.50: Il programma del Tic-Tac-Toe

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

0086	022E	A9 04		LDA #04	: MEMORIZZA IL PEZZO DEL
0087	0230	20 40 03		JSR UPDATE	: COMPUTER.
0088	0233	A9 0C		LDA #12	: ESEGUILO.
0089	0235	A0 07	WINTST	LDY #7	: CARICA STRUTTURA DI RICERCA
					: DI VITTORIA.
					: CICLA 7 VOLTE PER
					: CONTROLLARE LE SOMME DI
					: RIGA.
0090	0237	D9 2A 00	TSTLP	CMP ROWSUM, Y	: PER STRUTTURA VINCENTE.
0091	023A	F0 11		BEQ WIN	: VITTORIA SE STRUTTURA
					: TROVATA.
0092	023C	88		DEY	: CICLA E
0093	023D	10 F8		BPL TSTLP	: PROVA ANCORA.
0094	023F	A5 3A		LDA MOVNUM	: SE MOSSA NUMERO 9
0095	0241	C9 09		CMP #9	: ALLORA IL GIOCO È IN
					: PAREGGIO.
0096	0243	D0 CD		BNE PLAYLP	: ALTRIMENTI CONTINUA A
					: GIOCARE.
0097	0245	A9 FF		LDA # \$FF	: IMPOSTA IL LAMPEGGIO DI
					: TUTTE LE LUCI.
0098	0247	85 3D		STA LTMSKL	
0099	0249	85 3C		STA LTMSKH	
0100	024B	D0 4A		BNE DLY	: CONSERVA IL LAMPEGGIO.
0101	024D	C9 0C	WIN	CMP #12	: VITTORIA DEL COMPUTER?
0102	024F	F0 0E		BEQ INTDN	: SE SÌ, I.Q. PIU' BASSO.
0103	0251	A9 1E		LDA #30	: CARICA COSTANTE DI FREQ. ER
					: TONO VINCENTE
0104	0253	85 3F		STA FREQ	
0105	0255	A5 41		LDA INTEL	
0106	0257	C9 0F		CMP # \$0F	: I.Q. ERA PIU' ALTO POSSIBILE?
0107	0259	F0 0E		BEQ GTMSK	: SE SÌ, NON CAMBIARLO.
0108	025B	E6 41		INC INTEL	: AUMENTA I.Q.
0109	025D	D0 0A		BNE GTMSK	: ESEGUI IL LAMPEGGIO DELLA
					: RIGA.
0110	025F	A9 FF	INTDN	LDA # \$FF	: CARICA COSTANTE DI FREQ. PER
					: SUONO DI SCONFITTA.
0111	0261	85 3F		STA FREQ	
0112	0263	A5 41		LDA INTEL	: I.Q. = 0?
0113	0265	F0 02		BEQ GTMSK	: SE SÌ, NON DECREMENTARLO
0114	0267	C6 41		DEC INTEL	: I.Q. PIU' BASSO.
0115	0269	A9 00	GTMSK	LDA #0	: AZZERARE TUTTI I LED.
0116	026B	8D 01	A0	STA PORT1A	
0117	026E	8D 00	A0	STA PORT1B	
0118	0271	B6 00		LDX RWPT1, Y	: ACCETTA BIT
					: NELL'ACCUMULATORE PER
					: ACCENDERE
0119	0273				: I LED CORRISPONDENTI AL PRIMO QUADRATO
0120	0273				: NELLA RIGA VINCENTE.
0121	0273	20 6F 03		JSR LEDLTR	
0122	0276	B6 08		LDX RWPT2, Y	: ACCETTA IL SECONDO BIT.
0123	0278	20 6F 03		JSR LEDLTR	
0124	027B	B6 10		LDX RWPT3, Y	: ACCETTA IL TERZO BIT.
0125	027D	20 6F 03		JSR LEDLTR	
0126	0280	AD 01 00		LDA PORT1A	: MASCHERA I BIT NON NECESSARI
0127	0283	25 3D		AND LTMSKL	: DELLA MASCHERA DI LAMPEGGIO
0128	0285	85 3D		STA LTMSKL	
0129	0287	AD 00 A0		LDA PORT1B	
0130	128A	25 3C		AND LTMSKH	
0131	028C	85 3C		STA LTMSKH	
0132	028E	A9 FF		LDA # \$FF	: IMPOSTA LA DURATA DEL TONO
					: DI VINCITA/PERDITA.

Figura 11.50: Il programma del Tic-Tac-Toe (continua)



0133	0290	85 3E		STA DUR	
0134	0292	A5 3F		LDA FREQ	; ACCETTA LA FREQUENZA.
0135	0294	20 AD 00		JSR TONE	; ESEGUI IL TONO.
0136	0297	20 A4 03	DLY	JSR DELAY	; RITARDO PER MOSTRARE
					; VINCITA O PAREGGIO.
0137	029A	4C 04 02		JMP RESTRT	; INIZIO NUOVO GIOCO, NON
					; CAMBIARE I.Q.
0138	029D				
0139	029D				***** SUBROUTINE 'ANALYZE' *****
0140	029D				; ESEGUE UN'ANALISI STATICA DELLA MATRICE DEL GIOCO E
0141	029D				; RITORNA CON UNA MOSSA NEL REGISTRO X.
0142	029D				
0143	029D	A9 00		ANALYZ LDA #0	; PONI A 0 LA MASCHERA PER
					; ESEGUIRE MOSSE CASUALI
0144	029F	85 40		STA ODDMSK	; AI LATI.
0145	02A1	A9 08		LDA #8	; CONTROLLA PER MOSSA
0146	02A3	A2 03		LDX #03	; VINCENTE DEL COMPUTER.
0147	02A5	20 04 03		JSR FINDMV	
0148	02A8	D0 59		BNE DONE	; SE TROVATA, RITORNO.
0149	02AA	A9 02		LDA #2	; CONTROLLA PER MOSSA VINCENTE
0150	02AC	A2 03		LDX #3	; DEL GIOCATORE.
0151	02AE	20 04 03		JSR FINDMV	
0152	02B1	D0 50		BNE DONE	; SE TROVATA, RITORNO.
0153	02B3	A9 04		LDA #04	; IL COMPUTER PUO' IMPOSTARE
					; UNA TRAPPOLA?
0154	02B5	A2 02		LDX #2	
0155	02B7	20 04 03		JSR FINDMV	
0156	02BA	D0 47		BNE DONE	; SE SI, ESEGUILA.
0157	02BC	20 9A 00		JSR RANDOM	; ACCETTA UN NUMERO CASUALE...
0158	02BF	29 0F		AND #\$0F	; ...E RENDILO DA 0 A 15...
0159	02C1	C5 41		CMP INTEL	; PER L'IMPIEGO COME
					; DISCRIMINANTE STUPIDO
					; /INTELLIGENTE
0160	02C3	F0 02		BEQ OK	; SE SONO UGUALI, SALTA IL TEST.
0161	02C5	B0 2B		BCS RNDMV	; SE RND # > INTEL, ESEGUI UNA
					; MOSSA CASUALE.
0162	02C7	A6 3A	OK	LDX MOVNUM	
0163	02C9	E0 01		CPX #1	; PRIMA MOSSA?
0164	02CB	F0 25		BEQ RNDMV	; SE SI, GIOCA UN QUADRATO
					; QUALSIASI.
0165	02CD	E0 04		CPX #4	; QUARTA MOSSA?
0166	02CF	D0 0C		BNE TRAPCK	; SE NO, CONTINUA.
0167	02D1	A2 06		LDX #6	; CARICA L'INDICE ALLA PRIMA
					; SOMMA DI RIGA DIAGONALE.
0168	02D3	8A		TXA	; CARICA SOMMA DELLA
					; POSIZIONE DI RIGA AVENTE
					; C-G-C.
0169	02D4	D5 2A		CMP ROWSUM, X	; CONTROLLA SE LA PRIMA
					; DIAGONALE E' G-C-G.
0170	02D6	F0 16		BEQ ODDRND	; SE SI, GIOCA LATERALMENTE.
0171	02D8	E8		INX	; CONTROLLA LA SOMMA DI RIGA
					; NELLA DIAGONALE SUCCESSIVA
0172	02D9	D5 2A		CMP ROWSUM, X	
0173	02DB	F0 11		BEQ ODDRND	
0174	02DD	A9 01	TRAPCK	LDA #1	; IL GIOCATORE PUO' IMPOSTARE
					; UNA TRAPPOLA?
0175	02DF	A2 02		LDX #2	
0176	02E1	20 04 03		JSR FINDMV	
0177	02E4	D0 1D		BNE DONE	; SE SI, BLOCCA IL GIOCATORE.
0178	02E6	A6 1C		LDX GMBRD+4	; IL CENTRO E'

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

0179	02E8	D0 08		BNE RNDMV	: OCCUPATO?
0180	02EA	A2 05		LDX #5	: NO, GIOCALO.
0181	02EC	D0 15		BNE DONE	
0182	02EE	A9 01	ODDRND	LDA #1	: PONI ODDMASK AD 1, QUINDI
0183	02F0	85 40		STA ODDMSK	: LA MOSSA SARA' LATERALE.
0184	02F2	20 9A 00	RNDMV	JSR RANDOM	: ACCETTA # CASUALE PER LA
					: MOSSA.
0185	02F5	29 0F		AND # \$0F	: RENDILO DA 0 A 15.
0186	02F7	05 40		ORA ODDMSK	: ESEGUI # DISPARI SE RICHIESTO
					: ANGOLO.
0187	02F9	C9 09		CMP #9	: NUMERO TROPPO ALTO?
0188	02FB	B0 F5		BCS RNDMV	: SE SI, ACCETTANE UN ALTRO.
0189	02FD	AA		TAX	
0190	02FE	B5 18		LDA GMBRD, X	: SPAZIO OCCUPATO?
0191	0300	D0 F0		BNE RNDMV	: SE SI, ACCETTA UN'ALTRA
					: MOSSA.
0192	0302	E8		INX	: INCREMENTA X PER
					: UGUAGLIARE L'USCITA DI
					: FINDMV.
0193	0303	60	DONE	RTS	: RITORNO CON MOSSA IN X.
0194	0304				
0195	0304				: ***** SUBROUTINE 'FIND MOVE' *****
0196	0304				: TROVA UN QUADRATO CHE SODDISFA LE SPECIFICHE
0197	0304				: TRASFERITE IN A ED X.
0198	0304				: IL REGISTRO INDICE X CONTIENE
0199	0304				: LA MASCHERA CHE, SOTTOPOSTA AD OR CON IL
0200	0304				: NUMERO DI VOLTE, CHE UN QUADRATO APPROSSIMA LA
					: RIGA CON
0201	0304				: LA SOMMA DI RIGA NELL'ACCUM., DEVE PRODURRE UN UNO
0202	0304				: PER QUALIFICARE IL QUADRATO.
0203	0304				
0204	0304	86 39	FINDMV	STX TEMP2	: SALVA I REGISTRI.
0205	0306	85 38		STA TEMP1	
0206	0308	A9 00		LDA #0	: AZZERA I REGISTRI DI STATO DEL
					: QUADRATO.
0207	030A	A0 08		LDY #8	
0208	030C	99 21 00	CLRLP	STA SQSTAT, Y	
0209	030F	88		DEY	
0210	0310	10 FA		BPL CLRLP	
0211	0312	A0 07		LDY #7	: CICLA 7 VOLTE.
0212	0314	A5 38	CHEKLP	LDA TEMP1	: LA SOMMA DI RIGA E
0213	0316	D9 2A 00		CMP ROWSUM, Y	: UGUALE AL PARAMETRO?
0214	0319	D0 0F		BNE NOCHEK	: SE NO, PASSA ALLA SUCCESSIVA.
0215	031B	B6 00		LDX RWPT1, Y	: CONTROLLA IL PRIMO QUADRATO
					: NELLA RIGA.
0216	031D	20 39 03		JSR CNTSUB	: INCREMENTA IL SUO STATO SE
					: E VUOTO.
0217	0320	B6 08		LDX RWPT2, Y	: ESEGUI IL SECONDO QUADRATO.
0218	0322	20 39 03		JSR CNTSUB	
0219	0325	B6 10		LDX RWPT3, Y	: ED IL TERZO.
0220	0327	20 39 03		JSR CNTSUB	
0221	032A	88	NOCHEK	DEY	: PROVA LA RIGA SUCCESSIVA.
0222	032B	10 E7		BPL CHEKLP	
0223	032D	A2 09		LDX #9	
0224	032F	A5 39	FNMTCH	LDA TEMP2	: CARICA IL PARAMETRO...
0225	0331	35 20		AND SQSTAT-1, X	: (STATO DEL QUADRATO) AND
					: (PARAMETRO) - 0?
0226	0333	D0 03		BNE FOUND	: SE SI, ESEGUI X COME MOSSA.
0227	0335	CA		DEX	: INCREMENTA E PROVA IL
					: SUCCESSIVO
					: STATO DEL QUADRATO
0228	0336	D0 F7		BNE FNMTCH	
0229	0338	60	FOUND	RTS	

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

```

0230 0339      :
0231 0339      : ***** SUBROUTINE 'CONTSUB' *****
0232 0339      : INCREMENTA LO STATO DEL QUADRATO DEI QUADRATI
      : VUOTI.

0233 0339      :
0234 0339 B5 18 CNTSUB LDA GMBRD, X      : ACCETTA QUADRATO.
0235 033B D0 02      BNE NOCNT          : SE PIENO, SALTA.
0236 033D F6 21      INC SQSTAT, X      : INCREMENTA LO STATO DEL
      : QUADRATO.
0237 033F 80      NOCNT RTS              : FATTO.
0238 0340      :
0239 0340      : ***** SUBROUTINE 'UPDATE' *****
0240 0340      : ESEGUE LA MOSSA MEMORIZZANDO IL CODICE TRASFERITO
      : NELL'ACCUM.
0241 0340      : AL QUADRATO SPECIFICATO DAL REG. X.
0242 0340      : INOLTRE ACCENDE/LAMPEGGIA IL LED APPROPRIATO,
0243 0340      : E CALCOLA LE SOMME DI RIGA.
0244 0340      :
0245 0340 CA      UPDATE DEX              : DECREMENTA LA MOSSA PER
      : INDICIZZAZIONE
0246 0341 95 18      STA GMBRD, X      : ESEGUI LA MOSSA.
0247 0343 C9 04      CMP # $ 04        : MOSSA DEL COMPUTER?
0248 0345 F0 0D      BEQ NOBLNK        : SE SI, NON IMPOSTARE IL
      : LAMPEGGIO DEL LED.
0249 0347 20 98 03 JSR LIGHT          : MOSSA DEL GIOCATORE:
      : ACCETTA IL BIT
      : CORRISPONDENTE.
0250 034A      : AL LED DA LAMPEGGIARE
0251 034A 05 3D      ORA LTMSKL        : CARICA IL BIT NELLA MASCHERA
      : DI LAMPEGGIO.
0252 034C 85 3D      STA LTMSKL
0253 034E 90 04      BCC NOBLNK        : SE C = 0, NON IMPOSTARE IL BIT 9.
0254 0350 A9 01      LDA # 01          : PONI AD 1 IL BIT 9 PER IL
      : LAMPEGGIO.
0255 0352 85 3C      STA LTMSKH
0256 0354 20 6F 03 NOBLNK JSR LEDLTR   : ACCENDI IL LED.
0257 0357 A2 07      LDX # 7          : CICLO PER CALCOLARE LE
      : SOMME DI RIGA.
0258 0359 18      ADDROW CLC          : PREPARA PER L'ADDIZIONE.
0259 035A B4 00      LDY RWPT1, X      : ACCETTA L'INDIRIZZO DEL
      : PRIMO QUADRATO.
0260 035C B9 18 00      LDA GMBRD, Y      : ACCETTA I CONTENUTI DEL
      : QUADRATO.
0261 035F B4 08      LDY RWPT2, X      : SOMMA IL SECONDO QUADRATO
      : NELLA RIGA.
0262 0361 79 18 00      ADC GMBRD, Y
0263 0364 B4 10      LDY RWPT3, X      : SOMMA IL QUADRATO FINALE.
0264 0366 79 18 00      ADC GMBRD, Y
0265 0369 95 2A      STA ROWSUM, X      : SALVA LA SOMMA DI RIGA.
0266 036B CA      DEX
0267 036C 10 EB      BPL ADDROW        : ACCETTA LA SOMMA DI RIGA
      : SUCCESSIVA.
0268 036E 80      RST
0269 036F      :
0270 036F      : ***** SUBROUTINE 'LED LIGHTER' *****
0271 036F      : DATO UN ARGOMENTO NEL REGISTRO X, ACCENDE IL LED
0272 036F      : (0-8) CORRISPONDENTE A QUELL'ARGOMENTO.
0273 036F      :
0274 036F 20 98 03 LEDLTR JSR LIGHT    : ACCETTA IL BIT NELLA
      : POSIZIONE CORRETTA.
0275 0372 0D 01 A0      ORA PORT1A      : ACCENDI IL LED.
0276 0375 8D 01 A0      STA PORT1A

```

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

```

0277 0378 90 05          BCC LTRDN      ; SE IL LED # 9 NON ERA ACCESO,
                                ; SALTA.
0278 037A A9 01          LDA # 1        ; ACCENDI IL LED # 9.
0279 037C 8D 00 A0      STA PORT1B
0280 037F 60            LTRDN RTS        ; FATTO.
0281 0380              ;
0282 0380              ; ***** SUBROUTINE 'PLAYER'S MOVE' *****
0283 0380              ; ACCETTA LA MOSSA DEL GIOCATORE, CONTROLLANDO PER
                                ; ERRORI.
0284 0380              ;
0285 0380 A9 80          PLRMV LDA # $ 80 ; ESEGUE UN BEEP BREVE PER
                                ; SEGNALARE
0286 0382 85 3E          STA DUR         ; LA RICHIESTA D'INGRESSO DA
                                ; TASTIERA.
0287 0384 A9 10          LDA # $ 10
0288 0386 20 AD 00      JSR TONE
0289 0389 20 00 01 KEYIN JSR GETKEY      ; ACCETTA MOSSA.
0290 038C C9 0A          CMP # 10       ; FUORI LIMITI?
0291 038E B0 F9          BCS KEYIN      ; SE SI, ACCETTANE UN'ALTRA.
0292 0390 AA            TAX
0293 0391 F0 F6          BEQ KEYIN      ; SE MOSSA = 0, ACCETTANE
                                ; UN'ALTRA.
0294 0393 B5 17          LDA GMBRD-1, X ; QUADRATO VUOTO?
0295 0395 D0 F2          BNE KEYIN      ; SE NO, PROVA ANCORA.
0296 0397 60            RTS
0297 0398              ;
0298 0398              ; ***** SUBROUTINE 'LIGHT' *****
0299 0398              ; FA SCORRERE A SINISTRA UN BIT UNO NELL'ACCUMULATORE
0300 0398              ; FINO ALLA POSIZIONE CORRISPONDENTE
0301 0398              ; ALL'ARGOMENTO PASSATO NEL REGISTRO X, SE X=8
0302 0298              ; IL CARRY VIENE POSTO AD 1.
0303 0398              ;
0304 0398 86 38          LIGHT STX TEMP1 ; SALVA X
0305 039A A9 00          LDA # 0        ; AZZERA L'ACCUMULATORE PER
                                ; LO SCORRIMENTO.
0306 039C 38            SEC             ; ACCETTA IL BIT PER LO
                                ; SCORRIMENTO.
0307 039D 2A          SHIFT ROL A       ; SCORRIMENTO DEL BIT A
                                ; SINISTRA.
0308 039E CA            DEX
0309 039F 10 FC          BPL SHIFT      ; CONTO ALLA ROVESCIA E CICLO.
0310 03A1 A6 38          LDX TEMP1     ; RIMEMORIZZA X.
0311 03A3 60            RTS
0312 03A4              ;
0313 03A4              ; ***** SUBROUTINE 'DELAY' *****
0314 03A4              ;
0315 03A4 A0 FF          DELAY LDY # $ FF
0316 03A6 A2 FF          DL1 LDX # $ FF
0317 03A8 26 3E          DL2 ROL DUR    ; CONSUMA TEMPO.
0318 03AA 66 3E          ROR DUR
0319 03AC CA            DEX
0320 03AD D0 F9          BNE DL2
0321 03AF 88            DEY
0322 03B0 D0 F4          BNE DL1
0323 03B2 60            RTS
0324 03B3              ;
0325 03B3              ; ***** ROUTINE DI MANIPOLAZIONE INTERRUPT *****
0326 03B3              ; AD OGNI INTERRUPT, I LED LE CUI POSIZIONI NELLA
                                ; MASCHERA
0327 03B3              ; DI LAMPEGGIO CORRISPONDONO AD UNI, VENGONO
0328 03B3              ; ACCESSI E SPENTI ALTERNATIVAMENTE.
0329 03B3 48            INTVEC PHA

```

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

```

0330 03B4 AD 01 A0 LDA PORT1A
0331 03B7 45 3D EOR LTMSKL
0332 03B9 8D 01 A0 STA PORT1A
0333 03BC AD 00 A0 LDA PORT1B
0334 03BF 45 3C EOR LTMSKH
0335 03C1 8D 00 A0 STA PORT1B
0336 03C4 AD 04 A0 LDA T1LL
0337 03C7 68 PLA
0338 03C8 40 RTI
0339 03C9
0340 03C9 ; ***** SUBROUTINE 'INITIALIZE' *****
0341 03C9 ; INIZIALIZZA IL PROGRAMMA.
0342 03C9
0343 03C9 * - $ 50
0344 0050
0345 0050 A9 00 INIT LDA # 0 ; AZZERA LE MEMORIE.
0346 0052 A2 28 LDX # CLREND-CLRST
0347 0054 95 18 CLRALL STA CLRST, X
0348 0056 CA DEX
0349 0057 10 FB PBL CLRALL
0350 0059 AD 04 A0 LDA T1LL ; ACCETTA IL SEME DEL
; GENERATORE DI NUMERI
; CASUALI.

0351 005C 85 33 STA RNDSCR + 1
0352 005E 85 36 STA RNDSCR + 4
0353 0060 A9 FF LDA # $ FF
0354 0062 8D 03 A0 STA DDR1A ; IMPOSTA I/O
0355 0065 8D 02 A0 STA DDR1B
0356 0068 8D 02 AC STA DDR3B
0357 006B A9 00 LDA # 0 ; AZZERA I LED
0358 006D 8D 01 A0 STA PORT1A
0359 0070 8D 00 A0 STA PORT1B
0360 0073 ; PREDISPONI IL TIMER PER GLI INTERRUPT CHE
0361 0073 ; ESEGUONO IL LAMPEGGIO DEI LED.
0362 0073 20 86 8B JSR ACCESS ; LA MEMORIA NON PROTETTA
; DEL SISTEMA SYM-1
0363 0076 ; SERVE PER I VETTORI DI INTERRUPT.
0364 0076 A9 B3 LDA # : INTVEC ; CARICA IL BYTE BASSO DEL
; VETTORE DI INTERRUPT.
0365 0078 8D 7E A6 STA IRQVL ; MEMORIZZA ALLA LOCAZIONE
; DEL VETTORE DI INTERRUPT.
0366 007B A9 03 LDA # : INTVEC ; CARICA IL BYTE ALTO DEL
; VETTORE DI INTERRUPT.
0367 007D 8D 7F A6 STA IRQVH ; MEMORIZZALO.
0368 0080 A9 7F LDA # $ 7F ; AZZERA IL REGISTRO DI
; ABILITAZIONE INTERRUPT.
0369 0082 8D 0E A0 STA IER
0370 0085 A9 C0 LDA # $ C0 ; ABILITA INTERRUPT DEL TIMER 1.
0371 0087 8D 0E A0 STA IER
0372 008A A9 40 LDA # $ 40 ; ABILITA IL TIMER 1 NEL MODO A
; CORSA LIBERA.
0373 008C 8D 0B A0 STA ACR
0374 008F A9 FF LDA # $ FF
0375 0091 8D 04 A0 STA T1LL ; PONI LATCH BASSO SUL TIMER 1.
0376 0094 8D 05 A0 STA T1CH ; PONI LATCH ALTO & INIZIO
; CONT. INTERRUPT.
; ABILITA GLI INTERRUPT.
0377 0097 58 CLI
0378 0098 D8 CLD
0379 0099 60 RTS
0380 009A
0381 009A ; ***** SUBROUTINE 'RANDOM' *****
0382 009A ; GENERATORE DI NUMERI CASUALI: RESTITUISCE UN NUOVO
0383 009A ; NUMERO CASUALE NELL'ACCUMULATORE.

```

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

```

0384 009A          ;
0385 009A 38      RANDOMSEC
0386 009B A5 33   LDA RNDSCR + 1
0387 009D 65 36   ADC RNDSCR + 4
0388 009F 65 37   ADC RNDSCR + 5
0389 00A1 85 32   STA RNDSCR
0390 00A3 A2 04   LDX # 4
0391 00A5 85 32   RNDLP LDA RNDSCR, X
0392 00A7 95 33   STA RNDSCR + 1, X
0393 00A9 CA      DEX
0394 00AA 10 F9   BPL RNDLP
0395 00AC 60      RTS
0396 00AD          ;
0397 00AD          ; ***** SUBROUTINE 'TONE' *****
0398 00AD          ; GENERA UN SUONO: IL NUMERO DI 1/2 CICLI DEVE ESSERE
0399 00AD          ; IN DUR E LA
0400 00AD          ; COSTANTE DI LUNGHEZZA D'ONDA NELL'ACCUMULATORE.
0401 00AD          ;
0402 00AD 85 3F   TONE STA FREQ
0403 00AF A9 FF   LDA # $ FF
0404 00B1 8D 00 AC STA PORT3B
0405 00B4 A9 00   LDA # 00
0406 00B6 A6 3E   LDX DUR
0407 00B8 A4 3F   FL2 LDY FREQ
0408 00BA 88      FL1 DEY
0409 00BB 18      CLC
0410 00BC 90 00   BCC * + 2
0411 00BE D0 FA   BNE FL1
0412 00C0 49 FF   EOR # $ FF
0413 00C2 8D 00 AC STA PORT3B
0414 00C5 CA      DEX
0415 00C6 D0 F0   BNE FL2
0416 00C8 60      RTS
0417 00C9          . END

```

#### SYMBOL TABLE

SYMBOL      VALUE

ACCESS	8B86	ACR	A00B	ADDRESS	0359	ANALIZ	029D
CHEKLP	0314	CLRAL	0054	CLREND	0040	CLRLP	030C
CLRST	0018	CNTSUB	0339	COMPMV	0226	DDR1A	A003
DDR1B	A002	DDR3B	AC02	DELAY	03A4	DL1	A3A6
DL2	03A8	DL1	0297	DONE	0303	DUR	003E
FINDMV	0304	FL1	00BA	FL2	00B8	FNMITCH	032F
FOUND	0338	FREQ	003F	GETKEY	0100	GMBRD	0018
GTMSK	0269	IER	A00E	INIT	0050	INTDN	025F
INTEL	0041	INTVEC	03B3	IRQVH	A67F	IRQVL	A67E
KEYIN	0389	LEDLTR	036F	LIGHT	0398	LTMSKH	003C
LTMSKL	003D	LTRDN	037F	MOVNUM	003A	NOBLNK	0354
NOCHEK	032A	NOCNT	033F	ODDMSK	0040	ODDRND	02EE
OK	02C7	PLAYLP	0212	PLAYR	003B	PLRMV	0380
PORT1A	A001	PORT1B	A000	PORT3B	1C00	RANDOM	009A
RESTR	0204	RNDLP	00A5	RNDMV	02F2	RNDSCR	0032
ROWSUM	002A	RWPT1	0000	RWPT2	0008	RWPT3	0010
SHIFT	039D	SQSTAT	0021	START	0200	T1CH	A005
T1LL	A004	TEMP1	0038	TEMP2	0039	TONE	00AD
TRAPCK	02DD	TSTLP	0237	UPDATE	0340	WIN	024D
WINTST	0235						

END OF ASSEMBLY

Figura 11.50: Il programma del Tic-Tac-Toe (continua)

## APPENDICE B

### ISTRUZIONI IN ORDINE ALFABETICO DEL 6502

ADC	Somma con riporto	INC	Incrementa X
AND	AND Logico	INY	Incrementa Y
ASL	Spostamento Aritmetico a Sinistra	JMP	Salta
BCC	Opera diramazione se carry è zero	JSR	Salta alla subroutine
BCS	Opera diramazione se carry è uno	LDA	Carica l'accumulatore
BEQ	Opera diramazione se risultato = 0	LDX	Carica X
BIT	Verifica di bit	LDY	Carica Y
BMI	Opera diramazione se negativo	LSR	Spostamento logico a destra
BNE	Opera diramazione se diverso da 0	NOP	Non opera
BPL	Opera diramazione se positivo	ORA	OR Logico
BRK	Break	PHA	Introduce A
BVC	Opera diramazione se overflow è 0	PHP	Introduce lo stato P
BVS	Opera diramazione se overflow è 1	PLA	Estrae A
CLC	Azzerà carry	PLP	Estrae lo stato P
CLD	Azzerà il flag decimale	ROL	Rotazione a sinistra
CLI	Azzerà la disabilitazione interrupt	ROR	Rotazione a destra
CLV	Azzerà overflow	RTI	Ritorno da Interrupt
CMP	Confronta con l'accumulatore	RTS	Ritorno da subroutine
CPX	Confronta con X	SBC	Sottrae con riporto
CPY	Confronta con Y	SEC	Pone carry ad 1
DEC	Decrementa la memoria	SED	Pone decimale ad 1
DEX	Decrementa X	SEI	Pone disabilitazione interrupt ad 1
DEY	Decrementa Y	STA	Immagazzina l'accumulatore
EOR	OR Esclusivo	STX	Immagazzina X
		STY	Immagazzina Y
		TAX	Trasferisce A in X
		TAY	Trasferisce A in Y
		TSX	Trasferisce SP in X
		TXA	Trasferisce X in A
		TXS	Trasferisce X in SP
		TYA	Trasferisce Y in A

# APPENDICE B

## SET DI ISTRUZIONI DEL 6502: ESADECIMALE E TIMING

MNEMONICO		IMPLICATO			ACCUM.			ASSOLUTO			PAGINA ZERO			IMMEDIATO			ABS X			ABS Y		
		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ADC	(1)							6D	4	3	65	3	2	69	2	2	7D	4	3	79	4	3
AND	(1)							2D	4	3	25	3	2	29	2	2	3D	4	3	39	4	3
ASL					QA	2	1	OE	6	3	06	5	2				1E	7	3			
BCC	(2)																					
BCS	(2)																					
BEQ	(2)							2C	4	3	24	3	2									
BIT	(2)																					
BMI	(2)																					
BNE	(2)																					
BPL	(2)																					
BPK		00	7	1																		
BVC	(2)																					
BVS	(2)																					
CLC		18	2	1																		
CLO		D8	2	1																		
CLI		58	2	1																		
CLV		B8	2	1																		
CMP								CD	4	3	C5	3	2	C9	2	2	DD	4	3	D9	4	3
CPX								EC	4	3	E4	3	2	EO	2	2						
CPY								CC	4	3	C4	3	2	CO	2	2						
DEC								CE	6	3	C6	5	2				DE	7	3			
DEX		CA	2	1																		
DEY		B8	2	1																		
EOR	(1)							4D	4	3	45	3	2	49	2	2	5D	4	3	59	4	3
INC								EE	6	3	E6	5	2				FE	7	3			

INX		E8	2	1																		
INY		CB	2	1																		
JMP								4C	3	3												
JSR								20	6	3												
LDA	(1)							AD	4	3	A5	3	2	A9	2	2	BD	4	3	B9	4	3
LDX	(1)							AE	4	3	A6	3	2	A2	2	2						
LDY	(1)							AC	4	3	A4	3	2	AD	2	2	BC	4	3	BE	4	3
LSR					4A	2	1	4E	6	3	46	5	2				SE	7	3			
NOP		EA	2	1																		
ORA								00	4	3	05	3	2	09	2	2	1D	4	3	19	4	3
PHA		48	3	1																		
PHP		08	3	1																		
PLA		68	4	1																		
PLP		28	4	1																		
ROL					2A	2	1	2E	6	3	26	5	2				3E	7	3			
ROR					6A	2	1	6E	6	3	66	5	2				7E	7	3			
RTI		40	6	1																		
RTS		60	6	1																		
SBC	(1)							ED	4	3	E5	3	2	E9	2	2	FD	4	3	F9	4	3
SEC		38	2	1																		
SED		F8	2	1																		
SET		78	2	1																		
STA								8D	4	3	85	2					9D	5	3	99	5	3
STX								BE	4	3	86	2										
STY								BC	4	3	84	2										
TAX		AA	2	1																		
TAY		A8	2	1																		
TSX		BA	2	1																		
TXA		9A	2	1																		
TXS		98	2	1																		
TYA																						

(1) Somma 1 ad n se si attraversa il limite della pagina



(IND. X)			(IND. Y)			PAGINA Z, X			RELATIVO			INDIRETTO			PAGINA Z, Y			CODICI DI STATO DEL PROCESSORE											
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N	V	B	D	I	Z	C	MNEMONIC				
61	6	2	71	5	2	73	4	2										•	•					•	•	ADC			
21	6	2	31	5	2	35	4	2										•						•	•	AND			
						16	6	2										•						•	•	ASL			
									90	2	2																BCC		
									BO	2	2																BDS		
									FO	2	2																BEQ		
									30	2	2							Mr Ms						•			BIL		
									DO	2	2																BMI		
									10	2	2																BNE		
																													BPL
									50	2	2											1	1					BRK	
									70	2	2																	BVC	
																													BVS
																													CLC
																								0	0			CID	
																													CLT
C1	6	2	D1	5	2	D5	4	2										0										CLV	
																													CMP
																													CPX
																													CPY
						D6	6	2																					DEC
41	6	2	51	5	2	55	4	2																				DEX	
						F6	6	2																					DEY
																													EOR
																													INC

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

(2) Somma 2 ad n se si salta entro la pagina  
Somma 3 ad n se si salta ad un'altra pagina



I giochi costituiscono il modo pratico e divertente per assimilare le tecniche di programmazione studiate ("Programmazione del 6502" e "Applicazioni del 6502"). Nel processo passo-passo di spiegazione dei giochi proposti, infatti, da quelli semplici a quelli estremamente complessi, da quelli passivi a quelli strategici, il lettore affinerà le tecniche di programmazione in Assembler, impiegherà diverse strutture dati e implementerà le tecniche di ingresso/uscita. In pratica, le stesse tecniche e gli stessi concetti che qualsiasi programmatore rigoroso segue per la progettazione di una soluzione programmata di un problema che può essere di controllo industriale come di applicazioni commerciali. Ogni gioco comporta le regole, le istruzioni, gli algoritmi e il programma (struttura dei dati, tecniche di programmazione e descrizione dettagliata). Ogni programma, attentamente collaudato, occupa meno di 1 K di memoria e, pur richiedendo l'impiego di una scheda SYM, può essere facilmente adattato a qualsiasi altro computer basato sul 6502.