

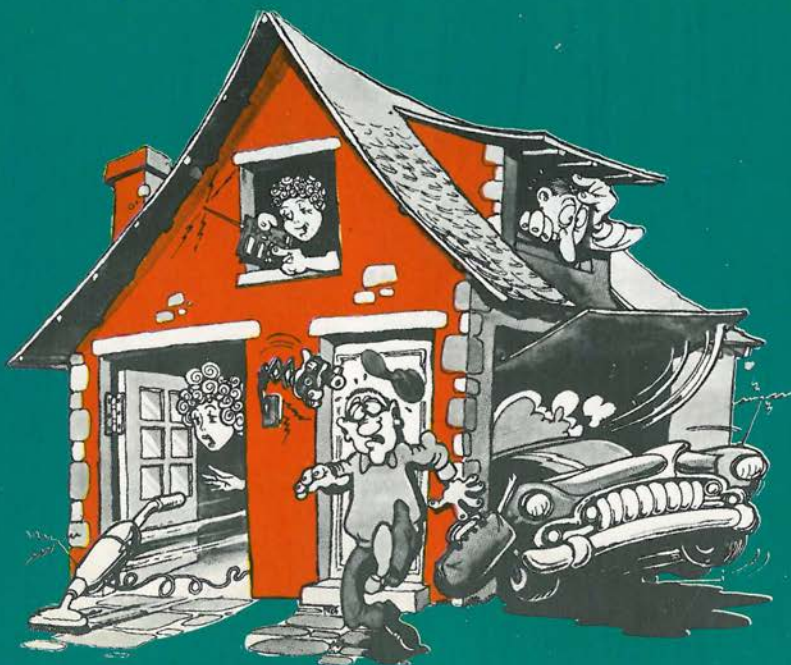
# APPLICAZIONI DEL 6502

EDIZIONE  
ITALIANA

Rodnay  
Zaks



GRUPPO  
EDITORIALE  
JACKSON





# **APPLICAZIONI DEL 6502**

di  
**Rodney  
Zaks**



**GRUPPO  
EDITORIALE  
JACKSON**  
Via Rosellini, 12  
20124 Milano

I programmi presentati in questo libro sono stati progettati per il loro valore educativo e controllati accuratamente. In ogni caso non si forniscono garanzie per qualunque motivo.

È stato compiuto ogni sforzo per fornire un'informazione completa ed accurata, comunque la Sybex non si assume responsabilità per quanto riguarda l'impiego; neppure qualsiasi infrazione di brevetti od altri diritti di parti terze che potrebbero derivare. Nessuna licenza è concessa dai costruttori mediante brevetto o diritti di brevetto. I costruttori si riservano il diritto di cambiare la circuiteria in qualsiasi momento senza preavviso.

In particolare sono soggetti a rapidi cambiamenti le caratteristiche tecniche ed i prezzi. Confronti e valutazioni sono presentati per il loro valore didattico e per criteri di scelta. Si rimanda il lettore ai dati del costruttore per caratteristiche più precise.

© Copyright per l'edizione originale SYBEX Inc. 1979

© Copyright per l'edizione italiana SYBEX Inc. 1981

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

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Fotocomposizione: Corponove s.n.c. - Bergamo

Stampa: Tipo Lito Ferrari Cesare & C. - Clusone (BG)



## RINGRAZIAMENTI

Molte sono le persone che hanno contribuito al controllo, sviluppo e miglioramento di questi programmi. Uno speciale riconoscimento dell'autore va a: Pierre Le Beux, Daniel David, Jaff Lin, Eric Martinot, Tricourt e ad Eric Novikoff (Assembler ASM65).

Le persone seguenti hanno inoltre contribuito ad apprezzabili commenti sulle bozze finali del manoscritto ed il loro contributo merita un notevole riconoscimento: John McClenoh, Doug Trusty, Phil Hooper, Daniel David, Robert Chitsum e John Smith.

Le seguenti industrie hanno consentito l'accesso ad informazioni apprezzabili, ed il loro contributo merita un notevole riconoscimento: Rockwell International, Synertek Systems, Apple.

I listing del Capitolo Quattro, parte 1, sono stati prodotti su un Sistema 65 della Rockwell. I listing della parte 2 sono stati prodotti con un assembler ASM65 riportato in Appendice A.

### Contributi Artistici:

Daniel Lenoury (Illustrazione di copertina)

Barry Janoff e Renate Woodbury

## RICHIAMO PER I PROGRAMMI

Nel corso del libro si incontreranno molti programmi pratici ed interessanti. Molti lettori vorranno sviluppare altri nuovi programmi di applicazione impiegando il 6502. Si prega di comunicarci eventuali miglioramenti dei programmi contenuti in questo libro oppure eventuali programmi sviluppati ex-novo purché ritenuti interessanti. Tutti i nostri libri vengono costantemente aggiornati ed ampliati. Si scriva all'autore al seguente indirizzo:

Rodnay Zaks, Ref. D302  
Sybex Inc.  
2020 Milvia Street  
Berkeley, CA 94704

È gradita anche la comunicazione di programmi addizionali eventualmente appropriati a questo libro per una sua successiva edizione.



## PREFAZIONE

Questo libro presenta le tecniche di applicazione pratica del microprocessore 6502. Si assume una conoscenza elementare della programmazione del microprocessore a livello del libro precedente di questa serie (Riferimento C202: Programmazione del 6502). La comprensione di come programmare il chip del microprocessore stesso (il 6502) è soltanto un presupposto per la programmazione effettiva di una scheda a microprocessore connessa a dispositivi reali. Il problema successivo consiste nell'imparare come scrivere i programmi di applicazione effettiva coinvolgendo le porte d'ingresso/uscita ed altre caratteristiche disponibili in un sistema reale. Questo libro vuole costituire un indirizzo a tale problema. Esso presenterà le tecniche ed i programmi richiesti per le applicazioni tipiche, impiegando i chip d'ingresso/uscita effettivamente disponibili su una scheda.

I programmi presentati in questo libro richiederanno un minimo di hardware effettivo per essere realizzati. Si raccomanda quindi agli utenti l'acquisizione di una certa pratica sui concetti e le tecniche qui presentate su un hardware reale. Verrà presentata una descrizione realistica delle schede delle applicazioni possibili. I programmi sono eseguibili su qualunque microcomputer su scheda basato sul 6502 quale il KIM, il SYM, l'AIM65 ed altri. Molti programmi possono essere eseguiti direttamente su una o più di queste schede, mentre altri richiederanno alcune varianti. In ogni caso i concetti e le tecniche sono comuni a tutti.

I programmi di applicazione presentati in questo libro consentiranno al lettore di realizzare un sistema completo di allarme per abitazione comprendente, tra l'altro, la rivelazione d'incendio, un pianoforte elettronico, un regolatore di velocità di un motore, un controllore di trenino, un orologio, un sistema simulante il controllo del traffico, un generatore di codice morse, un sistema industriale per il controllo della temperatura, comprendente una conversione da analogico a digitale ed altre applicazioni.

Questo libro si propone di illustrare tutte le tecniche richieste per le applicazioni reali del 6502. In questa serie sul 6502 è preceduto da "Programmazione del 6502" ed è seguito da "Giochi con il 6502".



# SOMMARIO

<b>RICHIAMO PER I PROGRAMMI</b> .....	III
<b>PREFAZIONE</b> .....	V
<b>CAPITOLO 1 – INTRODUZIONE</b> .....	1
<b>CAPITOLO 2 – I CHIP D'INGRESSO/USCITA</b> .....	5
Introduzione .....	5
Definizioni di base .....	5
Il 6520 (PIA) .....	8
Programmazione del 6522 .....	27
Il 6530 ROM-RAM I/O timer (RRIOT) .....	37
Il RIOT 6532 .....	38
Sommarlo .....	39
<b>CAPITOLO 3 – SISTEMI BASATI SUL 6502</b> .....	41
Introduzione .....	41
Un sistema 6502 «Standard» .....	41
Il KIM-1 .....	43
Il SYM-1 .....	46
L'AIM 65 .....	52
Altre schede .....	52
<b>CAPITOLO 4 – TECNICHE DI BASE</b> .....	53
Introduzione .....	53
<b>PARTE 1: Le tecniche</b> .....	55
Relè .....	55
Interruttori .....	61
Altoparlante .....	63
Un generatore Morse .....	64
Orologio .....	79
Un programma di controllo domestico .....	84
Combinatore telefonico .....	86
<b>PARTE 2: Combinazioni di tecniche</b> .....	93
Introduzione .....	93
Generazione del suono di una sirena .....	95
Rilevamento di un impulso di Ingresso .....	96
Misura dell'impulso .....	99
Un semplice programma musicale .....	102
Controllo del traffico con il KIM .....	105
Studio della tabella di moltiplicazione .....	109
Sommarlo .....	109

<b>CAPITOLO 5 — APPLICAZIONI INDUSTRIALI E DOMESTICHE</b>	<b>111</b>
Introduzione	111
Un sistema di controllo del traffico	116
LED a matrice di punti	124
Visualizzazione del valore espresso dagli interruttori	132
Generazione di tono	134
Musica	137
Un sistema di allarme antifurto	142
Controllo di un motore in corrente continua	145
Conversione analogico-digitale (un sensore di calore)	152
Sommario	161
<b>CAPITOLO 6 — LE PERIFERICHE</b>	<b>163</b>
Introduzione	163
Tastiera	163
Lettore del nastro di carta o tastiera ASCII	170
Microstampante	176
Sommario	181
<b>CAPITOLO 7 — CONCLUSIONI</b>	<b>183</b>
<b>APPENDICE A — UN ASSEMBLER IN BASIC DEL 6502</b>	<b>185</b>
Introduzione	185
Descrizione generale	185
Impiego dell'assembler	186
Sintassi	187
HP 2000F BASIC	188
<b>APPENDICE B — GIOCO DI MOLTIPLICAZIONE: IL PROGRAMMA</b>	<b>199</b>
<b>APPENDICE C — LISTING DEI PROGRAMMI (Capitolo 4, Parte 1)</b>	<b>202</b>
Programma 4-1: Morse	202
Programma 4-2: Tempo del giorno	206
Programma 4-3: Controllo domestico	208
Programma 4-4: Combinatore telefonico	210
<b>APPENDICE D — TABELLA DI CONVERSIONE ESADECIMALE</b>	<b>214</b>
<b>APPENDICE E — TABELLA DI CONVERSIONE ASCII</b>	<b>215</b>
<b>APPENDICE F — ISTRUZIONI DEL 6502 (in ordine alfabetico)</b>	<b>216</b>

## CAPITOLO 1

# INTRODUZIONE

Nell'apprendimento della programmazione, la comprensione del funzionamento del microprocessore stesso è soltanto il primo problema da risolvere. Questo problema è stato affrontato dal libro *Programmazione del 6502*. Il problema successivo consiste nell'imparare effettivamente a programmare, impiegando dispositivi d'ingresso/uscita connessi alla scheda del microprocessore. La risoluzione di tale problema è lo scopo di questo libro. Naturalmente è impossibile la realizzazione di un libro che copra tutti i dispositivi possibili. È quindi stata fatta una scelta tra i dispositivi più importanti che sono normalmente connessi ad un 6502 e vengono presentati i relativi programmi di applicazione ritenuti più generali.

Innanzitutto è necessario imparare a programmare effettivamente un PIO, il chip d'ingresso/uscita parallelo. Si imparerà l'impiego del polling e dell'interrupt. Si apprenderà come generare impulsi, ritardi di misura ed il controllo effettivo, dispositivi d'ingresso/uscita quali switch, relé, oppure dispositivi più complessi quali convertitori digitale - analogico, motori ed altri. Si imparerà inoltre come impiegare chip d'ingresso/uscita più complessi come un *temporizzatore programmabile*. Verranno presentate delle interfacce per semplici dispositivi per dare la possibilità di una realizzazione effettiva di una scheda di applicazione.

Per un effettivo apprendimento della programmazione si raccomanda vivamente una pratica diretta. Questo è l'unico modo per diventare un esperto programmatore. Per la pratica è necessario disporre di una scheda di microcomputer quali il KIM, il SYM, l'AIM65 oppure una scheda qualsiasi basata sul 6502. Poiché normalmente tutte le schede forniscono almeno un PIO (spesso 2) ed almeno due temporizzatori (talvolta di più), tutti i programmi presentati in questo libro possono essere trasferiti su queste schede con modifiche minime.

I Capitoli 4, 5 e 6 sono dedicati alla discussione sull'hardware addizionale richiesto per l'esecuzione di programmi specifici. In particolare i capitoli suddetti riportano la descrizione di schede di applicazione che possono essere realizzate a basso costo a partire da componenti molto comuni. Si suggerisce la realizzazione di tali schede per acquisire una certa pratica.

Questo non è comunque indispensabile. È possibile apprendere tutte le tecniche di base mediante la sola lettura del libro.

## Connessione del microprocessore al mondo esterno

La connessione del microprocessore stesso al mondo reale coinvolge preliminarmente la realizzazione di una scheda microprocessore di base, cui è demandata la connessione del dispositivo reale. Per la connessione del dispositivo reale sulla scheda sono richieste interfacce hardware e software. Questo libro presenta in dettaglio sia i componenti hardware sia i programmi richiesti per i dispositivi impiegati più comunemente. Per la realizzazione di programmi industriali che normalmente coinvolgono dispositivi costosi, quali i segnali di traffico, verranno impiegati dei dispositivi simulati sulla scheda applicativa, per esempio mediante dei LED. Se il programma deve essere applicato ad un sistema del traffico reale normalmente occorrerà cambiare soltanto l'interfaccia hardware. Il programma rimane sostanzialmente identico. Le

tecniche e gli artifici che si apprenderanno in questa sede saranno quindi applicabili alle situazioni reali.

## La pedagogia

Nel corso della lettura di questo libro si "impara eseguendo". Di ogni programma si presenterà in dettaglio: lo scopo, il diagramma di flusso, l'interfaccia hardware, i dispositivi, il programma stesso e l'analisi completa delle tecniche impiegate. Ogni capitolo è fondamentalmente autonomo. Per esempio non è necessaria la comprensione di tutte le caratteristiche del PIO, presentate al Capitolo 2, per la lettura del Capitolo 3. Per una comprensione completa si raccomanda comunque una lettura sequenziale. I contenuti del Capitolo 2 introducono tutti i più comuni chip di I/O parallelo impiegati in un sistema basato sul 6502, dal 6520 al 6532. Poiché tutte le schede esistenti del 6502 attualmente disponibili, impiegano questi chip standard, questo capitolo è fondamentale per coloro che non hanno familiarità con essi.

Il Capitolo 3 presenta la "Scheda Standard del 6502" ed alcune varianti ben note: KIM, SYM, AIM65 (ne esistono altre). La maggior parte degli esempi presentati in questo libro saranno eseguiti direttamente su un SYM e, con piccole variazioni su un KIM o su altre schede.

Il Capitolo 4 introduce le tecniche di applicazione di base per la connessione di dispositivi semplici: relé, switch, altoparlanti. La prima scheda di applicazione sarà impiegata per applicazioni che vanno da un generatore Morse ad un combinatore telefonico.

Il Capitolo 5 presenta applicazioni home ed industriali più complesse. La seconda scheda di applicazione sarà impiegata per applicazioni che vanno dal controllo simulato del traffico alla conversione analogico-digitale fino ad un sistema antifurto completo per la casa oppure un pianoforte elettronico.

Nel Capitolo 6 le periferiche a basso costo, attualmente disponibili, vengono connesse ad una scheda di microcomputer: dal lettore del nastro di carta alla tastiera e stampante.

Infine il Capitolo 7 presenta un sommario ed una sintesi.

Inoltre l'Appendice A riporta un assembler completo per il 6502 scritto in BASIC, per facilitare lo sviluppo di programmi complessi richiedenti un assembler.

Alla pagina successiva è riportato un modulo di programmazione standard per facilitare la scrittura dei programmi del 6502.







## CAPITOLO 2

# I CHIP D'INGRESSO-USCITA

### INTRODUZIONE

In questo libro collegheremo diversi dispositivi di ingresso-uscita a un 6502 con lo scopo di realizzare applicazioni pratiche con microcomputer. È quindi fondamentale conoscere le possibilità di ingresso-uscita di un sistema 6502. Il lettore che non ha familiarità con i termini o con le tecniche di base (ad esempio "polling") è invitato ad esaminarle nel precedente volume di questa serie *Programmazione del 6502*.

In questo capitolo esamineremo sistematicamente i chip di ingresso-uscita paralleli, usati su quasi tutti i sistemi 6502, per fornire le caratteristiche di ingresso-uscita richieste. È indispensabile sapere come lavora almeno un "PIO" prima di passare ai capitoli delle applicazioni. I dettagli sul funzionamento del timer o altre caratteristiche particolari (ad esempio uno shifter), non sono essenziali in una prima lettura e possono quindi essere omissi. Allo stesso modo i particolari ed i formati dei diversi registri interni del 6520, 6522, 6530, 6532, non sono importanti da ricordare. Essi sono forniti qui come riferimento per i capitoli successivi.

Suggeriamo comunque di leggere con attenzione almeno uno dei paragrafi su un PIO, per esempio il 6520 o il 6522, senza cercare di ricordare tutti i dettagli, ma insistendo sul loro modo di operare. Quasi ogni applicazione farà uso di un PIO, come avviene per i chip presentati in questo paragrafo.

Oltre a questi chip molti sistemi a microcomputer comprenderanno altre interfacce ingresso-uscita specializzate, come ad esempio un'interfaccia per cassette o per CRT. Chi è interessato ai dettagli di queste particolari interfacce, è rimandato alla documentazione fornita dai costruttori o al libro *Tecniche di Interfacciamento dei Microprocessori* sempre edito dal Gruppo Editoriale Jackson.

### DEFINIZIONE DI BASE

Questo paragrafo è un promemoria dei termini che useremo nel corso del capitolo.

Le tre principali realizzazioni ingresso-uscita su quasi tutti i sistemi a microcomputer sono il "PIO", l'"UART", e il "timer". Esaminiamole:

#### Il PIO

Il "PIO" o "chip d'Ingresso-uscita parallelo", è un componente che fornisce almeno due porte parallele ad 8 bit. In un PIO l'uso in direzione di ogni linea di ciascuna porta è di solito programmabile. La direzione di ogni linea è generalmente determinata dal contenuto di un "registro di direzione dati" associato ad ogni porta. Quando uno specifico bit del registro direzione dati è, per esempio, "0", la corrispondente linea della porta sarà un ingresso. Prima di usare il PIO il programmatore dovrà caricare il registro di direzione dati di ciascuna porta, per definire in quale direzione saranno usate le linee. Limitazioni particolari possono essere imposte dal

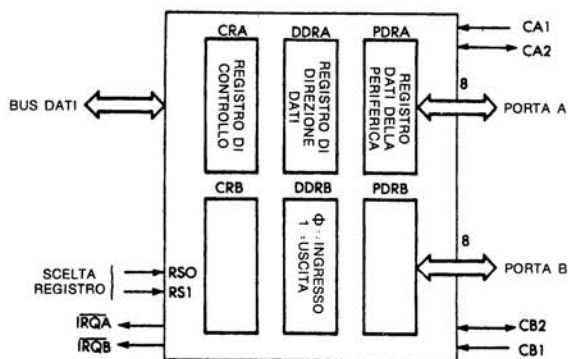


Figura 2-1: PIO tipico

costruttore, come ad esempio restringere la programmazione delle linee a gruppi di quattro, oppure assegnare funzioni speciali alle posizioni di alcuni bit, per esempio il bit sei e il bit sette. Alcune di queste limitazioni le vedremo nei chip presentati in questo capitolo. Il diagramma a blocchi dell'interno del "PIO standard" è mostrato in Figura 2-1. Sulla destra della figura si vedono i due buffer per la porta A e la porta B. Sulla sinistra dei buffer si vedono i registri di direzione dati associati a ciascuna porta. Inoltre questo diagramma semplificato è provvisto di due registri di controllo. Il registro di controllo è necessario per specificare la funzione dei segnali di controllo che sono forniti da questo PIO. In particolare esso deve determinare e controllare la procedura di "hand-shaking", oppure, ad esempio, può essere usato se i segnali di controllo forniscono il trigger per flag o interrupt e anche se avviene una transizione basso-alto, o una transizione alto-basso. In genere il programmatore chiarirà il contenuto del registro di controllo prima di fare un qualsiasi uso delle linee di controllo fornite dal componente. Il programmatore guarderà il contenuto del registro di controllo anche per determinare se è stato rilevato un interrupt interno o un'altra condizione speciale (informazione dello stato).

Oltre alle due porte dati, un PIO può fornire delle linee di controllo per permettere hand-shaking automatico con una periferica. Queste linee di controllo sono mostrate sulla destra del PIO standard di Figura 2-1, e sono chiamate rispettivamente CA1, CA2 per la porta A, CB1, CB2 per la porta B.

Come esempio di procedura di hand-shaking consideriamo una periferica esterna che fornisce un segnale di "DATI PRONTI" su CA1. Il microprocessore risponderà con un segnale di "RICHIESTA DATI" su CA2. Inoltre quando un segnale di "dati pronti" è ricevuto su CA1, può essere annotato nel registro di controllo, ed esternamente può essere generata una richiesta di interrupt per avvisare di questo evento il 6502. Questo è un semplice esempio tipico di sequenza di controllo, richiesta per un effettivo hand-shaking. Gran parte di questa procedura è automatica all'interno del PIO standard, e le opzioni sono definite dal contenuto del registro di controllo. I dettagli verranno mostrati per ogni PIO che descriveremo a partire da pagina 8.

## Il Timer

Una necessità di base in più applicazioni pratiche è la possibilità di generare particolari *ritardi*. I ritardi possono essere misurati con tecniche software o anche con timer hardware. Finché

non sono usati Interrupt nel sistema, i ritardi sono di solito generati, in maniera conveniente, mediante cicli software (per i particolari si veda il libro già menzionato *Programmazione del 6502*). Comunque, in situazioni più complesse, o in situazioni in cui possano esserci interrupt, è preferibile usare uno o più timer hardware esterni per generare o misurare ritardi fissati.

### *Impiego del Timer in Uscita*

Nella sua forma più semplice un timer hardware è un contatore equipaggiato con un registro (8 o 16 bit). Quando è usato in uscita il registro del timer è caricato con un valore fornito dal programma. È poi dato un segnale di "avanti" ed esso comincia a contare. Molti timer usano il clock di sistema, ma non necessariamente (di solito un MHz di clock = impulsi di un microsecondo). Il numero posto nel registro del contatore sarà decrementato di uno per ogni successivo impulso di clock. Se il valore nel registro era N, il contenuto del contatore raggiungerà zero dopo N impulsi, cioè dopo N microsecondi, considerando impulsi di un microsecondo. Quando il contatore arriva a zero, si posizionerà un flag di stato nel chip del timer e/o si genererà un interrupt esterno. In funzione della precisione richiesta il programma sonderà i dispositivi timer o accetterà Interrupt. In questo capitolo saranno presentati programmi tipici.

Se il timer sarà costituito da un solo registro a 8 bit, conterà solo da uno a 256. Il massimo ritardo con un clock standard, potrà essere solamente di 256 microsecondi. Questo ritardo è troppo corto per la maggior parte delle applicazioni. Naturalmente sarà possibile usare l'Interrupt generato alla fine dei 256 microsecondi, per aggiornare una locazione di memoria, poi sondare questa locazione di memoria per vedere se ha raggiunto uno specifico valore. Comunque ciò porterà ad una non accurata misurazione del tempo e ad una certa scomodità nel procedimento. Quindi un timer che impiega un registro ad 8 bit sarà insufficiente. Per superare questa limitazione sono usate due tecniche. Concettualmente la più semplice è quella di impiegare come contatore un registro a 16 bit. Il contatore può quindi contare da uno a 64 K, cioè da un microsecondo a 65 · 536 microsecondi, cioè circa 65 millisecondi. Questo è certamente sufficiente per la maggior parte delle applicazioni. Tale tecnica comunque richiede che il timer sia caricato con almeno due operazioni, dal momento che il bus dei dati può trasferire solamente 8 bit. Un inconveniente è che il programma deve prima caricare una metà del registro, poi l'altra metà. L'altra tecnica per generare ritardi su un ampio campo è di usare circuiti divisori interni al timer. Un timer siffatto apparirà al programmatore come un dispositivo formato, probabilmente, di quattro registri. Per esempio, se è usato il primo registro, il ritardo generato sarà espresso in unità di clock (in genere 1 microsecondo). Se è usato il secondo registro l'unità di ritardo sarà 8 volte il ciclo di clock; nel terzo l'unità di temporizzazione sarà 64 volte il ciclo di clock, e nel successivo sarà  $1 \cdot 024$  volte il ciclo di clock (o approssimativamente un millisecondo, assumendo il clock di 1 MHz. Questo approccio è molto più conveniente al programmatore e offre la possibilità di caricare il timer con una sola operazione, usandolo su un ampio campo. Aumenta comunque la complessità interna del dispositivo.

### *Impiego del Timer in Ingresso*

Un timer può essere usato in ingresso per misurare la durata di un impulso esterno, oppure il tempo trascorso fra due impulsi successivi. In questo caso il contenuto iniziale del contatore del timer è zero e il suo registro interno sarà incrementato ad ogni intervallo di tempo. Una volta che il ritardo è stato misurato sarà posizionato un flag dal dispositivo, oppure sarà generato un interrupt e il programma sarà responsabile della lettura del contenuto del registro del contatore, il quale indica la durata dell'evento esterno.

### *Treni di Impulsi*

Un timer può essere usato non solo per generare o misurare un impulso, ma anche per generare o contare un treno di impulsi. Quando si genera un ritardo o si misura un impulso, il mo-

do di funzionamento è in genere detto "one-shot" (conta-uno). Quando si genera un treno di impulsi si dice modo "free-running" (corsa libera). Oltre a queste, esistono diverse possibilità per decidere se far partire o fermare il timer in corrispondenza di una transizione alto-basso o basso-alto del segnale, o anche se devono essere considerati i livelli invece degli impulsi. Inoltre possono essere specificati la temporizzazione e il valore logico del flag di interrupt. Oltre a ciò possono essere programmate le condizioni secondo cui lo stato interno è posizionato. A causa del grande numero di possibili variazioni, ogni timer tende ad essere fortemente personalizzato, ed è necessario studiarlo in dettaglio prima di farne uso.

## L'UART

"UART" sta per "Universal Asynchronous Receiver-Transceiver" (Ricetrasmittitore Asincrono Universale). La funzione essenziale dell'UART è eseguire conversioni seriale-parallelo e parallelo-seriale. Oltre a ciò l'UART standard fornisce un numero di opzioni abitualmente richieste per le comunicazioni seriali con dispositivi esterni, ad esempio la parità (controllo, inibizione o generazione) e bit d'inizio e fine. La conversione è eseguita da uno shifter interno. Un tale shifter può anche essere incorporato in alcuni chip ingresso-uscita.

## Dispositivi reali d'ingresso-uscita del 6502

Praticamente ogni scheda basata sul 6502 richiederà almeno 2 PIO ed un timer. Queste funzioni saranno in genere eseguite da una combinazione di chip 6520 e 6530 o da un 6522 e 6532. I 6520 e 6530, che saranno descritti in seguito, sono gli originali chip ingresso-uscita che furono introdotti dalla MOS Technology. Il 6502 è ora prodotto da parecchi altri costruttori, ad esempio Synertek e Rockwell, e sono stati introdotti altri chip addizionali di supporto, ad esempio il 6522 e il 6532; in futuro ne saranno probabilmente introdotti altri.

In questo momento, comunque, i chip più importanti sono il 6520, 6530, 6522 e il 6532. Ora descriveremo questi quattro fondamentali chip d'ingresso-uscita.

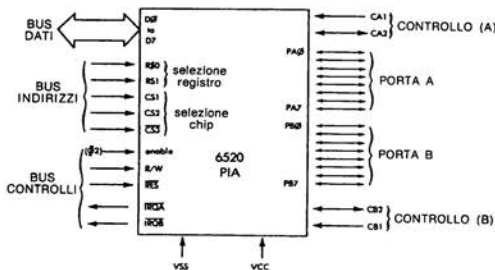


Figura 2-2: Il PIA 6520

## IL 6520 (PIA)

Il 6520 è un "PIO" quasi puro, per come lo abbiamo definito. Esso è stato progettato come una sostituzione pin-for-pin del Motorola M6820, ed è stato chiamato dal costruttore "peripheral interface adapter" o "PIA" (adattatore d'interfaccia della periferica). I segnali del 6520 sono mostrati in Figura 2-2. La sua architettura interna è mostrata in Figura 2-3.

Con riferimento alla Figura 2-3, si può vedere che questo dispositivo fornisce due porte parallele d'ingresso-uscita, la porta A e la porta B. Ogni porta è fornita di un buffer. Ad ogni modo le due porte non sono completamente identiche e in realtà il buffer lavora soltanto come buffer d'uscita e non d'ingresso. Un registro di direzione dati ("DDR") è disponibile per ogni porta e specifica la direzione di ogni linea della porta. Un valore "0" in DDR indica un ingresso e un valore "1" indica un'uscita. La scelta della convenzione deriva da una considerazione di sicurezza: quando si applica un "RESET", il contenuto di tutti i registri va a zero ed il registro di direzione dati diventerà zero. Come risultato tutte le linee saranno configurate come ingressi; questo è un modo sicuro per avviare un sistema. Nessun impulso esterno può infatti essere generato fino a che il programma non comincia ad essere eseguito.

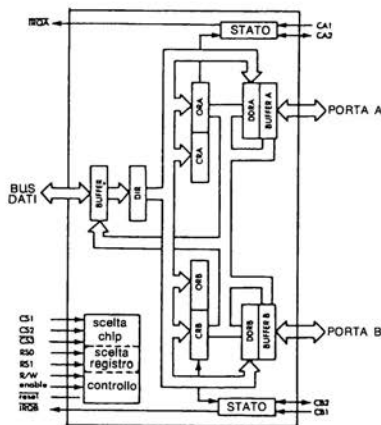


Figura 2-3: Architettura Interna del 6520

Inoltre ogni porta è equipaggiata con due registri, il *registro di controllo* ed il *registro d'uscita*. I dati spediti dal 6502 al dispositivo sono raccolti nel *registro d'uscita* (ORA) della porta indicata, dove sono trattenuti. La funzione del *registro di controllo* (CRA) sarà spiegata in seguito. Esso specifica il ruolo di diverse possibilità di controllo e contiene l'informazione dello stato, per ogni porta.

Infine ogni porta è munita di due linee di controllo esterno, indicate con CA1 e CA2 per la porta A. CA1 è una linea unidirezionale dal dispositivo al 6520. CA2 è una linea bidirezionale, che può essere usata come ingresso o come uscita.

Le due porte sono logicamente equivalenti e simmetriche, come indicato in Figura 2-3. Ad ogni modo esistono delle differenze pratiche. In particolare, la capacità di guida della porta B è superiore a quella della porta A, ed il ruolo dei segnali di controllo non è completamente simmetrico.

Guardando ora alla sinistra della Fig. 2-3, o alla Figura 2-2, il bus dati, collega il buffer interno del 6520 al bus dati del sistema. Possono essere generate dal dispositivo due richieste di interrupt, se ciò è specificato dai contenuti dei registri di controllo delle porte A e B; esse sono rispettivamente IRQA e IRQB. Infine tre ingressi di selezione chip devono essere specificati per il dispositivo e sono indicati con CS1, CS2 e CS3. Questo progetto era usato dalla Motorola per permettere l'adeguato collegamento diretto di 8 separati dispositivi al bus dati, senza aver bi-

sogno di un decodificatore esterno di indirizzi. In pratica l'alto numero di ingressi di selezione chip, può risultare un inconveniente che sarà indicato più avanti (mancanza di un selezione registro). Sono forniti due ingressi di selezione registro, connessi al bus indirizzi. Sono indicati con RS0 e RS1. Questo significa che il dispositivo 6520, appare al programmatore come *quattro* locazioni di memoria. Ciò può sorprendere dal momento che abbiamo appena visto (vedi Figura 2-3) che ci sono quattro registri per porta, cioè un totale di *otto* registri. Come è possibile indirizzare 8 registri con soli quattro valori? Questo è un problema generato dalla limitazione del numero di pin del dispositivo. Un bit del registro di controllo, il bit 2, è usato per individuare i due set di registri. Quando il bit 2 del registro di controllo è uguale a "0", è selezionata la direzione dati per quella porta. Quando è a "1", è selezionato il buffer di interfaccia della periferica.

Infine, sono disponibili tre ulteriori linee di controllo: "R/W" (lettura o scrittura), "enable" (di solito la fase due del clock), ed infine "reset".

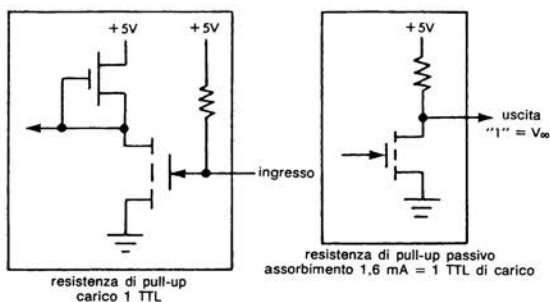


Figura 2-4: Buffer A

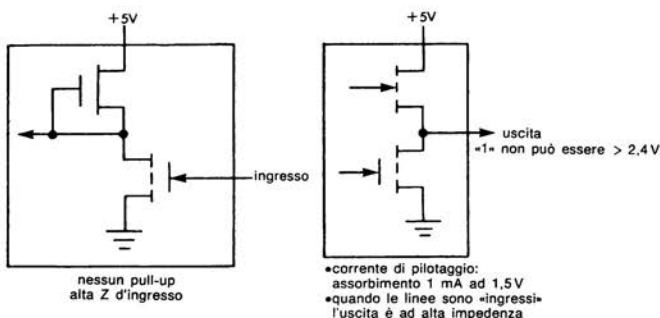


Figura 2-5: Buffer B



### Differenze fra la porta A e la porta B

La porta A e la porta B, anche se logicamente equivalenti, sono fisicamente diverse. I buffer della porta A usano *pull-up passivi*. Essi possono assorbire 1,6 mA, mettendo in grado i buffer di pilotare un carico TTL standard. Sulla porta B i buffer sono costituiti da dispositivi in *push-pull* (vedi Figura 2-4 e Figura 2-5). Dal momento che essi sono dispositivi *attivi*, la tensione logica di "1" non può essere superiore a 2,4 Volt ( $V_{DD}$  nel caso della porta A). Comunque essi hanno una corrente di pilotaggio superiore (1 mA a 1,5 V), possono quindi essere collegati direttamente a LED, o a Darlington. Infine, quando la porta B è usata come *ingresso*, il buffer d'uscita, entra in uno stato di *alta-impedenza*, l'ingresso presenterà quindi un'alta impedenza (maggiore di un MOhm). I particolari del buffer della porta A sono mostrati in Figura 2-4, e quelli del buffer della porta B in Figura 2-5.

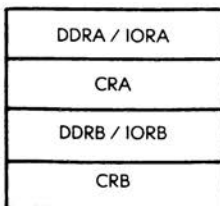


Figura 2-6: Mappa di Memoria del 6520

### I Registri Interni

Andiamo ora a considerare in maggior dettaglio le caratteristiche del 6520. Innanzi tutto, come abbiamo già notato, il 6520 è munito di sei registri interni: i due buffer (che condividono l'indirizzo del registro d'uscita), i due registri di direzione dati ed i due registri di controllo. Comunque, a causa della limitazione del numero di pin, sul dispositivo sono disponibili solamente due pin per la selezione registro, indicati rispettivamente con RS0 e RS1. La conseguente mappa di memoria del 6520 è mostrata in Figura 2-6. Essa mostra ad esempio che i registri DDRA e IORA condividono lo stesso indirizzo logico di memoria. Il registro di controllo è indirizzato indipendentemente. Il 6520 distingue internamente il DDRA e IORA tramite il valore del bit 2 del registro di controllo. La selezione dei registri è presentata in Figura 2-7. Quando il bit del registro di controllo è "0" viene selezionato il DDR. Quando è a "1" è selezionato il registro 10 o registro buffer. Il registro di controllo è l'unico che può essere indirizzato direttamente da RS0 e RS1, dal momento che è necessario definire il contenuto di tale registro prima di accedere agli altri.

RS1	RS0	CRA-2	CRB-2	REGISTRO SELEZIONATO
0	0	1	-	BUFFER A
0	0	0	-	DDRA
0	1	-	-	CRA
1	0	-	1	BUFFER B
1	0	-	0	DDRB
1	1	-	-	CRB

Figura 2-7: Selezione di Registro nel 6520

Questo schema implica che l'inizializzazione del dispositivo è un po' più complessa di quel che dovrebbe essere, e che, se il programma ha necessità di accedere successivamente al DDRA e al IORA, devono essere inserite ogni volta istruzioni per modificare il contenuto del bit 2 del CRA. Questo è certamente un inconveniente.

## II Registro di Controllo

Ciò che è contenuto nel registro di controllo è mostrato in Fig. 2-8. Si è già dichiarato che il bit 2 di questo registro ha una funzione speciale: esso individua i registri DDR e IOR per una data porta. Gli altri bit all'interno del registro, permettono opzioni per le due linee di controllo disponibili su ogni porta, e due bit sono riservati per lo stato o l'informazione di interrupt. Le funzioni del registro di controllo A, sono regolate dai bit 3, 4 e 5 e sono mostrate in Figura 2-9.

7	6	5	4	3	2	1	0
IRQ1	IRQ2	Controllo CA/B2		Scelta DDRA/B	Controllo CA B1		

Figura 2-8: Registri di Controllo del 6520

CRA BIT			MODO	EFFETTO
5	4	3		
1	0	0	Handshake in lettura	<ul style="list-style-type: none"> <li>• Transizione di interrupt sull'ingresso CA1 alza CA2.</li> <li>• Un'istruzione di lettura sulla Porta A abbassa CA2.</li> </ul>
1	0	1	Impulso d'uscita	<ul style="list-style-type: none"> <li>• La lettura sulla Porta A abbassa CA2 per un ciclo (= dispositivo riconosciuto).</li> </ul>
1	1	0	Uscita manuale	abbassa CA2
1	1	1	Uscita manuale	alza CA2

Figura 2-9: Controllo CA2 del 6520

CRA BIT			MODO	EFFETTO
5	4	3		
1	0	0	Handshake in scrittura	<ul style="list-style-type: none"> <li>• Transizione di interrupt sull'ingresso CB1 alza CB2.</li> <li>• Scrittura dati sulla Porta B abbassa CB2.</li> </ul>
1	0	1	Impulso d'uscita	<ul style="list-style-type: none"> <li>• Scrittura dati sulla Porta B abbassa CB2 per un ciclo (= dispositivo riconosciuto).</li> </ul>
1	1	0	Uscita manuale	abbassa CB2
1	1	1	Uscita manuale	alza CB2

Figura 2-10: Controllo CB2 del 6520

Le funzioni delle due linee di controllo della porta B, sono regolate dai bit 3, 4 e 5 del suo registro di controllo e sono mostrate in Figura 2-10. I bit 0 e 1 eseguono il controllo dell'interrupt per gli ingressi CA1 e CB1. Essi sono mostrati in Figura 2-11.

CR BIT		TRANSIZIONE ATTIVA DEL SEGNALE D'INGRESSO	USCITA I/O
0	0	negativo	disabilita (alto)
0	1	negativo	abilita (va basso quando il bit 7 di CRA è alzato da una transizione di CA1 / CB1)
1	0	positivo	disabilita (alto)
1	1	positivo	abilita (come sopra)

Figura 2-11: Controllo Interrupt (Ingressi CA1, CB1)

## Impiego del 6520

Dopo un "RESET" il contenuto di tutti i registri sarà zero. Il 6520 deve, quindi, innanzi tutto essere inizializzato per definire la configurazione di ingresso ed uscita per entrambe le sue porte. Devono anche essere specificate le opzioni per il registro di controllo e normalmente il 6520 viene fatto partire con un "1" nel bit 2 di tale registro, cosicché il 6502 può accedere direttamente al registro IOR.

Una sequenza tipica è la seguente:

```

LDA    #$0F      "00001111" = 4 INGRESSI, 4 USCITE
STA    DDRA      CONFIGURA LA DIREZIONE
LDA    #CONTROL  OPZIONI DI CONTROLLO:
                  BIT 2 = 1 PER INDIRIZZARE IORA
STA    CRA
```

### Ingresso - Uscita

Si può mandare fuori dati sulla porta A con le seguenti due istruzioni (assumendo il bit 2CRA = "1"):

```

LDA    #DATI      O ANCHE LDA $ 20 (DA MEMORIA)
STA    IORA
```

La lettura di un Ingresso collegato al 6520 avviene tramite le istruzioni:

```

LDA    IORA
STA    $20        SALVA I DATI IN MEMORIA
```

In questo caso abbiamo caricato in modo immediato il contenuto dell'accumulatore nella locazione di memoria 20 (esadecimale). Ad ogni modo non è indispensabile usare questo metodo. In molti casi si vuole semplicemente portare il contenuto di IORA nell'accumulatore e poi, probabilmente, se ne vuole valutare il valore ma non necessariamente si vuole memorizzarlo.

## Avvertenze sul 6520

Oltre a ricordare la differenza fra le porte A e B, bisogna anche ricordare alcune caratteristiche particolari delle funzioni di controllo. In particolare, sulla porta A o B i bit 6 e 7 sono azzerati se il 6 è un ingresso e se stiamo leggendo. Inoltre la lettura di un dato sulla porta B, azzerava

il bit 7. L'handshake su CB2, a differenza dell'handshake su CA2, serve per la *scrittura* di dati su B (CA2 agisce come *read* o *write*). Infine il bit 6 o 7 possono generare interrupt.

## Polling del 6520

Il modo più semplice per fare il polling di più 6520, è di controllare lo stato dei bit 6 e 7 del registro di controllo. Quando entrambi questi bit sono a zero, il dispositivo non richiede alcun servizio. Se uno dei due bit è a "1", è stato generato un interrupt interno e deve essere servito.

### Prima Tecnica

Per individuare facilmente quale dei quattro dispositivi ha richiesto servizio, può essere usato il metodo di accesso sequenziale a una tabella, purché gli indirizzi dei quattro dispositivi siano sequenziali in memoria. L'indirizzo n sarà assegnato a CRA1, l'n + 1 a CRB1, l'n + 2 a CRA2, l'n + 3 a CRB3, etc. Il programma dovrà poi fare uso del metodo di indirizzamento indiretto indicizzato come mostrato di seguito:

START	LDX	#8	INDICE
NEXT	LDA	(BASE-1,X)	ACCESSO AL SUCCESSIVO CR
	BMI	SERVICE	IRQ È ALTO?
	DEX		X = X - 1
	BEQ	START	
	BNE	NEXT	

BASE	-WORD CRA 1	PIO #1	PORT A
	-WORD CRB 1		PORT B
	-WORD CRA 2	PIO #2	PORT A
	-WORD CRB 2		PORT A
	-WORD CRA 3	PIO #3	PORT A
	-WORD CRB 3		PORT B
	-WORD CRA 4	PIO #4	PORT A
	-WORD CRB 4		PORT B

Figura 2-12: Identificazione del PIO

Il registro indice è inizialmente posto al valore "8" ed è successivamente decrementato di 1 ogni volta che si esegue il ciclo di polling. L'accumulatore è caricato con il contenuto dell'ultimo elemento entrato nella tabella, in precedenza:

```
LDA      (BASE-1,X)
```

Se il bit 7 era al valore logico alto (il bit 7 è il bit del segno 0 flag "N"), dovrà essere eseguita la routine di servizio:

```
BMI      SERVICE
```

Se il flag N non era stato posto ad 1, X viene decrementato ed è valutato il successivo CR:

```
DEX
BEQ      START      RICOMINCIA SE X = 0
BNE      NEXT       AVANTI SE X NON 0
```

*Miglioramento: scambiando le ultime due istruzioni si aumenta la velocità del programma?*

## Seconda Tecnica

All'interno di ogni CRA devono essere valutati due bit: il 6 e il 7. L'istruzione "BIT" del 6502 è stata creata per questo scopo specifico. Essa esegue un confronto non distruttivo che valuta il contenuto dei bit 6 e 7. Il programma di polling del 6520 appare in Figura 2-13.

BIT	CRA		
	BMI	IRQA7	
	BVC	NOTA1	
IRQA6	...		TROVATO IRQ A2 (BIT 6)
.			
.			
IRQA7			TROVATO IRQ A1 (BIT 7)
.			
.			
NOTA1	BIT	CRB	LO STESSO PER LA PORTA B
	BMI	IRQB7	
	BVC	NEXT2	
IRQB6	...		TROVATO IRQ B2 (BIT 6)
.			
.			
IRQB7			TROVATO IRQ B1 (BIT 7)
.			
.			
NEXT2	BIT		SUCCESSIVO 6520
	.		
	.		

Figura 2-13: Identificazione delle Porte

L'istruzione "BIT" è usata per valutare se uno dei due bit, 6 o 7, è a "1". Ciò si esegue con:

BIT	CRA	
	BMI	IRQA7
	BVC	NOTA1
	BIT 7 = 1	
	NON È STATO TROVATO INTERRUPT	

Se nessun flag era stato posto ad 1 si passerà a NOT A1, dove sarà valutato il CRB. Il bit 7 è pesato con l'istruzione BMI. Se il bit 7 era uno, sarà posto ad 1 il bit del segno N, e sarà eseguita la routine all'indirizzo IRQA7.

In caso contrario sarebbe stato il bit 6 ad essere posto ad 1 e, sarebbe stata eseguita dopo BMI, la routine all'indirizzo IRQ6.

Questa procedura può essere applicata ad ognuno dei 6520. Si noti che questo modo di procedere assegna la priorità maggiore ad A7 poi viene A6.

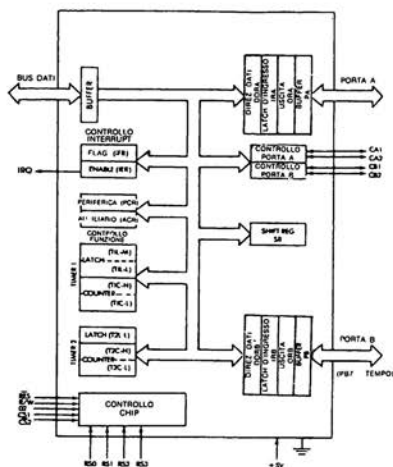


Figura 2-14: Architettura Interna del 6522

## Il 6522

Il 6522, Introdotto dalla MOS Technology, e prodotto a. che da Rockwell International e Synertek, è il successore del 6520.

Il chip 6522, chiamato VIA (Versatile Interface Adapter: Adattatore d'Interfaccia Versatile), è una combinazione PIO-timer-shifter. Internamente è costituito da 16 registri che sono mostrati in Figura 2-14. La corrispondente mappa di memoria è in Figura 2-15.

Possono essere distinti quattro gruppi di registri in base alla loro funzione:

1. I registri PIO (indirizzi da 0 a 3, più indirizzo F).
2. I registri timer (due timer, indirizzi da 4 a 9).
3. Il registro di shift (Indirizzo A).
4. I registri di controllo (indirizzi da B a E).

Questi quattro gruppi saranno ora esaminati in dettaglio per spiegare le capacità del 6522.

## La Sezione PIO

La sezione PIO fornisce due porte bidirezionali ad 8 bit. Ogni porta è costituita da un registro d'ingresso/uscita. Sono chiamati rispettivamente ORA e ORB per la porta A e la porta B. Sono mostrati in Figura 2-14. Ognuno di essi è associato ad un registro di direzione, rispettivamente DDRA e DDRB. Quando il corrispondente bit nel dato del registro di direzione è uguale ad "1" la linea collegata all'OR sarà un'uscita. Quando il bit di direzione dati è "0" la linea corrispondente sarà un'ingresso. La polarità è stata scelta in modo che tutte le linee siano ingressi quando viene applicato un "reset".

In questo PIO esiste un'asimmetria: la porta A è costituita da due registri OR, con e senza la possibilità di handshake.

00	ORB (PB0 FINO A PB7)	dati I/O, porta A
01	ORA (PA0 FINO A PA7)	usato per controllo-influenze handshake
02	DDR B	registri 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	latch basso
09	T2C-H	contatore alto
0A	SR	shift register
0B	ACR	ausiliario
0C	PCR (CA1, CA2, CB2, CB1)	periferica
0D	IFR	flag
0E	IER	enable
0F	ORA	Registro d'uscita A (non influenza handshake)

Figura 2-15: Mappa di Memoria del VIA 6522

P51	P52	P51	P50
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

00	ORB	+ handshake	I/O PARALLELO
01	ORA		
02	DDR B		
03	DDR A		
04	T1L-L/T1C-L (R)	- aziona flag ms 01 (R) - T1C-L, T1L-L, - aziona Flag ms 01 (R)	TIMER 11
05	T1C-H/T1L-H + T1C-H (W)		
06	T1L-L		
07	T1L-H		
08	T2L-L/T2C-L (R)	- aziona flag ms 02 (R) - T2C-L, T2L-L, - aziona Flag ms 02 (R)	TIMER 12
09	T2C-H		
0A	SR		
0B	ACR		
0C	PCR		CONTROLLO
0D	IFR		
0E	IER		
0F	ORA		

Figura 2-16: Registri del 6522

## Impiego del PIO

Prima di usare il PIO, bisogna caricare un opportuno valore nei registri di direzione dati, per

dare la configurazione corrispondente di Ingresso o uscita ai bit dell'I/O. Come esempio, abbiamo scelto la porta A come uscita e la porta B come ingresso.

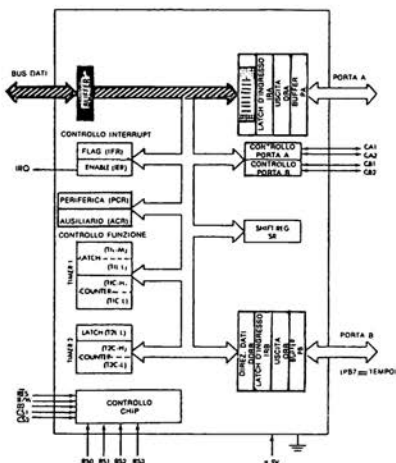


Figura 2-17: Impiego del 6522: STA DDRA

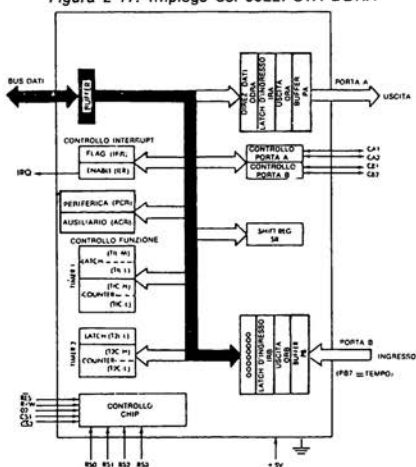


Figura 2-18: Impiego del 6522: STA DDRB



```

LDA    #$FF      "11111111" = USCITA
STA    DDRA
LDA    #0
STA    DDRB      B è un INGRESSO

```

(vedi Figura 2-17 e 2-18)

Ora portiamo fuori il valore "00000001" sulla porta A (vedi Figura 2-19):

```

LDA    #$01      "00000001"
STA    ORA

```

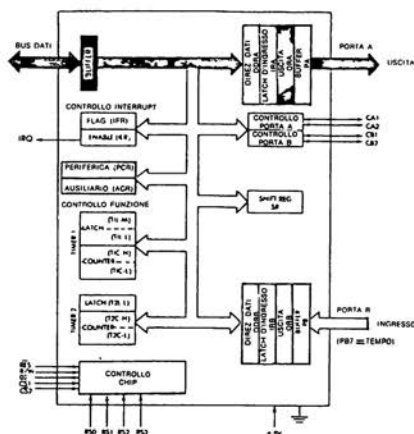


Figura 2-19: Impiego del 6522: STA ORA

Infine leggiamo il valore della porta B portandolo in accumulatore (vedi Figura 2-20).

```

LDA    ORB

```

Quando si usano i registri OR, è necessario, di solito, controllare un segnale di *stato* per esser certi che il dispositivo con cui si sta parlando sia *pronto* ad ascoltare o a trasmettere. Questo è detto *handshaking*. Le operazioni dei segnali di controllo, richiesti per eseguire quanto detto, sono spiegate di seguito.

## I Due Segnali di Controllo (Registro di Controllo della Periferica)

Ogni porta è fornita di due linee di controllo, chiamate CA1, CA2 e CB1, CB2 (vedi Figura 2-14, la parte destra). Per esempio, prima di spedire dati ad una stampante, ad esempio una te-

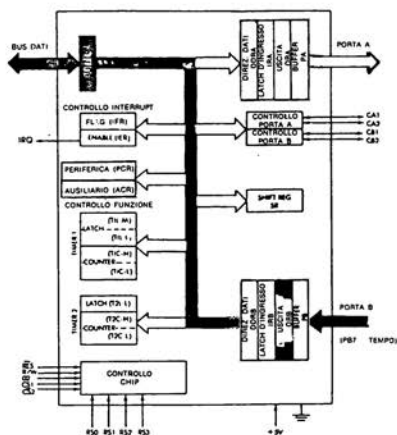


Figura 2-20: Impiego del 6522: LDA ORB

letype, il microprocessore deve accertarsi che questa non sia occupata e sia pronta ad accettare il carattere successivo. Questo lo si farà tramite una procedura di *handshaking*.

Quando la stampante non è più occupata, è pronta per ricevere il carattere successivo e manderà un impulso o una transizione di livello al 6522. Questa transizione di livello, o impulso, deve essere rilevata, conservata dal dispositivo e controllata poi dal programma. Il segnale sarà trasmesso su uno dei due ingressi di controllo, CA1 o CB1.

Il 6522 permette una grande flessibilità nel chiarire la natura dei segnali in ingresso o uscita.

Si può decidere se porre ad 1 il flag di interrupt interno deve essere una transizione *alto-a-basso* (o "negativa"), (fronte di discesa), o una transizione *basso-a-alto* (o "positiva") (fronte di salita). Questo è specificato dal bit 0 (per CA1) e il bit 4 (per CB1) del *registro di controllo della periferica* (PCR). "0" corrisponde alla transizione alto-a-basso, "1" corrisponde alla transizione basso-a-alto (vedi Figura 2-21).

Una volta chiarita la natura del segnale diventa possibile valutarlo.

*Valutazione dello stato:* è possibile rilevare se è avvenuta una transizione controllando il

7	6	5	4	3	2	1	0
CB2 controllo		CB1 controllo		CA2 controllo		CA1 controllo	

Figura 2-21: Registro di Controllo di Periferica

7	6	5	4	3	2	1	0
IRQ(R) EN(W)	T1	T2	CB1	CB2	SR	CA1	CA2

Figura 2-22: Registro Abilitazione Flag Interrupt (IFR/IER)

CR BIT	TRANSIZIONE ATTIVA DEL SEGNALE D'INGRESSO	USCITA IRQ
0 0	negativo	disabilita (alto)
0 1	negativo	abilita (va basso quando il bit 7 di CRA è alzato da una transizione di CA1 CB1)
1 0	positivo	disabilita (alto)
1 1	positivo	abilita (come sopra)

Figura 2-23: Funzione Linee di Controllo (ACR)

contenuto dei bit 1 o 4 (rispettivamente per CA1 e CB1) del *registro interrupt-flag (IFR)* (vedi Figura 2-22). Questo bit sarà "0" se non è stato ricevuto alcun segnale, e diventerà "1" una volta rilevata la transizione appropriata. Dopo la lettura dello stato "1", deve essere possibile eseguire il *reset* del bit, in modo che si possa procedere al rilevamento dell'evento successivo. Questo sarà eseguito scrivendo un "1" nella posizione opportuna del registro, oppure anche leggendo, o scrivendo, il corrispondente registro d'ingresso-uscita.

PCR3	PCR2	PCR1	Modo
0	0	0	Modo Fronte Negativo Interrupt CA2 (azzerà IFRO ORA) – Una transizione negativa del segnale d'ingresso, pone ad 1 il flag di interrupt CA2 (IFRO). IFRO si azzerà sulla lettura o scrittura del Registro d'Uscita della Periferica A, o scrivendo il valore logico 1 nel IFRO.
0	0	0	Modo Fronte Negativo Interrupt CA2 (azzerà IFRO) – Si posiziona IFRO, su una transizione del segnale d'ingresso CA2. La lettura o scrittura del ORA, non azzerà il flag di interrupt CA2. IFRO si azzerà scrivendo il valore logico 1 nel IFRO.
0	1	0	Modo Fronte Positivo Interrupt CA2 (azzerà IFRO ORA) – Pone ad 1 il flag di interrupt CA2 su una transizione positiva del segnale d'ingresso CA2. Azzerà IFRO con la lettura o scrittura del Registro d'Uscita della Periferica A.
0	1	1	Modo Fronte Positivo Interrupt CA2 (azzerà IFRO) – Posiziona IFRO su una transizione positiva del segnale d'ingresso di CA2. Lettura o scrittura del ORA, non azzerano il flag di interrupt di CA2. IFRO si azzerà scrivendo il valore logico 1 nel IFRO.
1	0	0	Modo d'Uscita Handshake CA2 – L'uscita CA2 va bassa, su una lettura o scrittura del Registro d'Uscita della Periferica A. CA2 ritorna alto, con una transizione attiva su CA1.
1	0	1	Modo d'Uscita Impulso CA2 – CA2 va basso per un ciclo, in seguito alla lettura o scrittura del Registro d'Uscita della Periferica A.
1	1	0	Modo Uscita Basso CA2 – In questo modo l'uscita CA2 è mantenuta bassa.
1	1	1	Modo Uscita Alta CA2 – In questo modo l'uscita CA2 è mantenuta bassa.

Figura 2-24: Operazioni Dettagliate del PCR (concessione Rockwell)

PCR7	PCR6	PCR5	Modo
0	0	0	Modo Fronte Negativo Interrupt CA2 (azzerà IFR3 ORB) – Pone ad 1 il flag di interrupt di CB2 (IFR3), su una transizione negativa, del segnale di ingresso di CB2. Azzerà IFR3, su una lettura o scrittura del Registro d'Uscita Periferico B, o scrivendo il valore logico 1 nel IFR3.
0	0	1	Modo Fronte Negativo Interrupt CB2 (azzerà IFR3) – Posiziona IFR3, su una transizione negativa del segnale d'ingresso di CB2. La lettura o scrittura del ORB, non azzerà il flag di interrupt. IFR3 si azzerà scrivendo il valore logico 1 in IFR3.
0	1	0	Modo Fronte Positivo Interrupt CB2 (azzerà IFR3 ORB) – Posiziona il segnale d'ingresso di CB2. Azzerà il flag di interrupt di CB2, su una lettura o scrittura del ORB, oppure scrivendo il valore logico 1 in IFR3.
0	1	1	Modo Fronte Positivo Interrupt CB2 (azzerà IFR3) – Posiziona IFR3, su una transizione positiva del segnale d'ingresso di CB2. Una lettura o scrittura del ORB non azzerà il flag di interrupt di CB2. IFR3 si azzerà scrivendo il valore logico 1 in IFR3.
1	0	0	Modo d'uscita Handshake CB2 – Abbassa CB2 su una operazione di scrittura del ORB. Riporta CB2 alto con una transizione attiva del segnale d'ingresso di CB1.
1	0	1	Modo d'uscita Impulso CB2 – Mantiene CB2 basso per un ciclo in seguito a un'operazione di scrittura del ORB.
1	1	0	Modo Uscita Basso Manuale CB2 – In questo modo l'uscita CB2 è mantenuta bassa.
1	1	1	Modo Uscita Alta Manuale CB2 – In questo modo l'uscita CB2 è mantenuta alta.

Figura 2-25: Operazioni Dettagliate del PCR

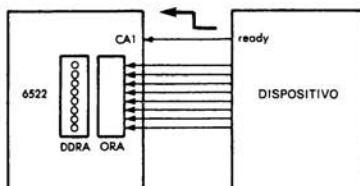


Figura 2-26: Lettura Dati Quando c'è il Ready

## Un Semplice Esempio di Ingresso

Consideriamo una transizione basso-a-alto del segnale di "ready" dalla periferica e una configurazione di Ingresso sulla porta A (vedi Figura 2-26). Quando i dati sono pronti vengono letti nell'accumulatore. Il programma è il seguente:

```

LDA      #0
STA      DDRA      PONI GLI INGRESSI
LDA      #1
STA      PCR        INTERRUPT CA1 DA BASSO-A-ALTO
LDA      IFR        LEGGI I FLAG DI INT
WAIT     AND      #$02 00000010 MASCHERA IL BIT PER CA1
        BEQ      WAIT  PRONTO?
        LDA      ORA    LEGGI I DATI IN INGRESSO

```

Miglioramento: Si possono modificare le due istruzioni "LDA IFR AND#\$02" per migliorare l'efficienza?

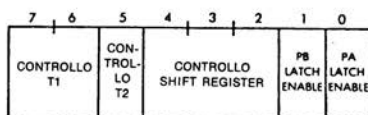


Figura 2-27: Registro Controllo Ausiliario — 6522

## Latching d'Ingresso/Uscita

L'ingresso e l'uscita del 6522, non sono simmetrici. Le uscite sono sempre *latched*. Ciò avviene per mezzo del registro d'ingresso/uscita chiamato OR (registro d'uscita). Gli ingressi non sono necessariamente *latched*. Questo è specificato tramite i bit "0" e "1" (rispettivamente porta A e porta B) del registro ausiliario di controllo (ACR). Quando questi bit valgono "0", non si ha alcun latching sull'ingresso. Quando questi bit valgono "1", gli ingressi sono latching (vedi Figura 2-27). Quando l'ingresso non è latched significa che il programma sta leggendo il valore delle linee d'ingresso collegate alla porta che è letta attualmente. Quando gli ingressi sono latched il latch è abilitato dalla transizione attiva di CA1 o CB1, a seconda della porta usata. Il valore è poi conservato nel registro latch fino a che non si è ricevuto l'impulso successivo sulla linea di controllo. *Attenzione:* in uscita il programma legge i comandi di latch, che possono essere o no uguali al contenuto dell'OR.

## Invio di un Segnale di Controllo in Uscita

CA2 o CB2 sono usati per fornire uno strobe di controllo (vedi Figura 2-14). Dal momento che queste linee sono bidirezionali, possono assumere la configurazione di uscite ponendo ad "1" rispettivamente il bit 3 o 7 del registro di controllo periferico (per A2 o B2) (vedi Figura 2-24).

La natura del segnale può essere specificata dall'uno o dall'altro dei due livelli oppure da un impulso. Il valore "0" rispettivamente nei bit 2 o 6 (per A o B), corrisponde ad un impulso. Il valore "1" corrisponde ad un livello. Quando si è deciso un livello è possibile scegliere un valore positivo o uno negativo. Questo è ottenuto ponendo ad "1" o a "0" rispettivamente i bit 1 e 5 (per A2 e B2) (vedi Figura 2-24).

Infine, quando si genera un impulso, la sua durata può essere controllata con i bit 1 e 5 (rispettivamente per A2 e B2) del registro di controllo.

Quando il bit è posto uguale a "0", si genera un singolo ciclo di strobe. Quando questo bit è posto uguale a "1", sarà generato un impulso d'uscita, che rimarrà basso dall'istante in cui si accede al registro OR (lettura o scrittura), fino alla successiva transizione del segnale su CA1 o CB1.

## Sommario dell'Uscita di Controllo

Praticamente può essere considerato un impulso di qualunque durata e polarità. Esso può essere usato per osservare un dispositivo esterno (per interrogarlo), per acconsentire ad un trasferimento di dati ad un altro dispositivo collegato alla stessa linea, o per controllare lo stato del dispositivo (on, off o altre possibilità).

Un sommario dei bit del registro di controllo periferico è mostrato in Figura 2-21, e i dettagli sono mostrati in Figura 2-24 e 2-25.

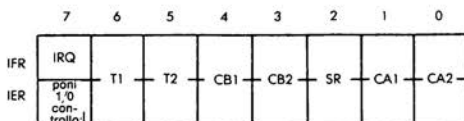


Figura 2-28: Registri di Interrupt

## Interrupt

Gli Interrupt sono controllati da due registri, il registro di abilitazione dell'interrupt (IER), e il registro flag di interrupt (IFR). I registri sono mostrati in Figura 2-28. Essi condividono lo stesso indirizzo di memoria. Uno è un registro d'Ingresso e l'altro è un registro d'uscita.

Il registro del flag di interrupt è un registro d'ingresso. Ogni bit da 0 a 7 sarà posto ad 1 quando è rilevato un interrupt su ognuna delle linee esterne (CA1, CA2, CB1, CB2), sul registro di shift (SR), su ognuno dei due timer (T1 e T2). Il bit 7 è alzato quando tutti gli altri bit del registro sono alti.

Il registro di abilitazione dell'interrupt (IER), abiliterà e disabiliterà tutti gli interrupt. Ogni bit di IER corrisponde a un bit di IFR (vedi Figura 2-28). Quando un bit ha il valore "0", il corrispondente interrupt è *disabilitato* e non sarà inviato. Quando un bit è al valore "1", l'interrupt è *abilitato* e se si presenta viene memorizzato. Diviene quindi possibile, per il programma, leggere il contenuto del registro IFR e controllare ogni bit per determinare se si è presentato un interrupt. Per posizionare convenientemente ogni bit di IER, è usato il bit 7 di IER congiuntamente a un segnale di lettura o scrittura ed il contenuto del bus dei dati è quindi copiato nel registro IER. Se il bit 7 di IER è "0", ogni "1" cancellerà un flag di abilitazione. Se il bit 7 è "1", ogni "1" scritto in IER alzerà un'abilitazione.

ESEMPIO: Abilitiamo gli interrupt CA1 e CA2 e disabilitiamo tutti gli altri (vedi Figura 2-28):

```

LDA    #$7C      «01111100» = CANCELLA I BIT DA 2 A 6
STA    IER
LDA    #$83      "10000011" = ABILITA I BIT 0 E 1
STA    IER

```

ESERCIZIO 2-1: Scrivere un programma che abiliti l'interrupt CA1 e disabiliti gli altri.

ESERCIZIO 2-2: Disabilita CB1 e CB2 lasciando gli altri invariati.

## Identificazione dell'Interrupt

Quando possono avvenire più Interrupt contemporaneamente, cioè quando sono usati parecchi bit di IFR, il programma dovrà controllare il contenuto di IFR e determinare quale interrupt è avvenuto. L'ordine in cui controlla questi bit determinerà la priorità dell'interrupt corrispondente. Per esempio, se l'interrupt da T1 ha la priorità più alta, questo è il bit che dovrà essere controllato per primo. Il modo più semplice per controllare il contenuto di IFR, è di farlo scorrere a destra o a sinistra di un bit e controllare il valore del bit che esce (nel Carry) facendo un test sul carry. Questa tecnica assegna priorità, in un modo da destra-a-sinistra o da sinistra-a-destra, ai segnali di Figura 2-28.

ESERCIZIO 2-3: Consideriamo la Figura 2-28. Elencate i dispositivi in ordine di priorità effettiva, considerando che il contenuto di IFR è fatto scorrere a sinistra dal programma di controllo.

Naturalmente è anche possibile fare il controllo per combinazioni di interrupt, controllando i

valori di particolari bit del registro IFR. Per maggiori dettagli sugli interrupt e polling, consultare il Capitolo 3 di "Programmazione del 6502".

## I Timer

Il 6522 è munito di due *timer interni*. Questi due timer possono essere usati come ingresso od uscite.

Usato in *uscita* un timer può generare un segnale o un treno di impulsi.

Il timer usato in *ingresso*, può misurare la durata di un impulso oppure contare il numero di impulsi ricevuti. Quando genera o misura la durata di un impulso, si dice che il timer funziona in modo "*one-shot*". Sia il timer 1 che il timer 2 possono essere usati in questo modo.

Quando il timer è usato per generare o contare un treno continuo di impulsi si dice che funziona in "*free-running mode*". Solo il timer 1 può essere usato in questo modo.

Prima di usare il timer in uscita, bisogna caricare il registro del contatore con un valore: per la generazione di impulsi il contatore dovrà contenere il numero di impulsi di clock da generare, oppure la durata dell'impulso usando il timer in ingresso, il suo registro deve essere azzerato. Quando conta impulsi esso conterrà il numero di impulsi contati fino a quell'istante. Quando si usa per rilevare un impulso esso conterrà la sua durata.

### Confronto fra il Timer 1 e il Timer 2

Il timer 2 può essere usato in ingresso per contare gli impulsi applicati a PB6 di IORB (vedi Figura 2-14). Quando invece è usato in uscita esso può generare solamente un impulso di durata stabilita su PB6, non può generare un treno di impulsi. L'uno o l'altro di questi due tipi di funzionamento è selezionato con il bit 7 del registro di controllo ausiliario (ACR) (vedi Figura 2-27). Il valore "0" corrisponde al modo one-shot e "1" al modo pulse-counting.

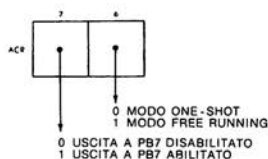


Figura 2-29: 6522: Controllo dei Modi di T1 con il Registro Controllo Ausiliario

Il timer 1 è diverso dal timer 2 e offre maggiori possibilità. Esso ha quattro modi di funzionamento che sono illustrati in Figura 2-29. Può funzionare nel modo one-shot o nel modo free-running. Inoltre può abilitare o disabilitare un'uscita su PB7. Il modo di funzionamento è selezionato con il bit 6 del registro di controllo ausiliario. Tale bit è al valore "0" per il modo one-shot e a "1" per il modo free-running.

Il bit 7 dice se PB7 è abilitato oppure no: con il valore "0" PB7 non è abilitato, con "1" è abilitato (vedi Figura 2-30).

## Caricamento dei Contatori

Ogni timer impiega un contatore a 16 bit. La parte inferiore di tale contatore, deve essere caricata prima di quella superiore. Caricando la parte superiore del contatore, si azzer automaticamente il flag di interrupt e il timer comincia a funzionare. Il timer 1 è fornito anche di un vero latch a 16 bit, mentre il timer 2 ne è sprovvisto. Ciò consente al timer 1 di funzionare continuamente nel modo "free-running"; il latch è trasferito automaticamente al contatore quando

ACR7 ABILITAZIONE USCITA	ACR 6 ABILITAZIONE FREE RUN	MODO
0	0 (ONE-SHOT)	Genera INT di time out quando è caricato T1 PB7 disabilitato.
0	1 (FREE RUN)	Genera INT continui PB7 disabilitato.
1	0 (ONE-SHOT)	Genera INT e impulso d'uscita su PB7 ogni volta che è caricato T1. one-shot e larghezza d'impulso programmabile.
1	1 (FREE RUN)	Genera INT continui e uscita ad onda quadra su PB7.

Figura 2-30: 6522 – Selezione Modi Operativi del Timer 1 con il Registro Controllo Ausiliario

questo raggiunge zero. Nel timer 1 i valori dei latch possono essere letti o scritti senza influenzare i contatori. Tale fatto è sfruttato per generare forme d'onda della complessità voluta. I particolari sull'indirizzamento del timer sono illustrati in Figura 2-31.

	INDIRIZZO	SCRITTURA	LETTURA
TIMER 1	-- 04	T1L-L	T1C-L/ - azzerà T1 - ME di T1
	-- 05	T1L-H + T1C-H + T1L-L → T1C-L - azzerà T1 - ME di T1	T1C-H
	-- 06	T1L-L	T1L-L
	-- 07	T1L-H - azzerà flag ME di T1	T1L-H
TIMER 2	-- 08	T2L-L	T2C-C - azzerà flag ME di T2
	-- 09	T2C-H T2L-L → T2CL azzerà flag ME di T2	T2C-H

Figura 2-31: Indirizzamento del Timer

## Durata Reale

Consideriamo per il timer 1, la forma d'onda mostrata in Figura 2-32. Si noti che la durata reale è data dal valore del conteggio ("N") più 2 o più 1,5. Per ottenere quindi una maggiore precisione nella temporizzazione, l'utilizzatore dovrà caricare nel registro del contatore il numero desiderato di periodi meno 2.



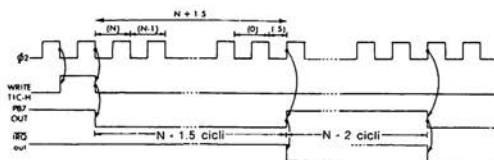


Figura 2-32: Timer 1 in Modo Free Running

## II Registro di Shift

Tale registro serve per la conversione seriale-parallelo e parallelo-seriale. La velocità di shift può essere controllata da tre temporizzazioni diverse: il timer 2, la fase 2 del clock ( $\phi_2$ ) e un clock esterno. Quando la temporizzazione è presa dall'esterno, ciò è segnalato con i bit 2 e 3 del registro di controllo ausiliario (vedi Figura 2-27). Il bit 4 di tale registro specifica l'ingresso o l'uscita. La tabella completa che illustra la funzione di questi bit è riportata in Figura 2-33.

ACR4	ACR3	ACR2	Modo
0	0	0	Registro shift disabilitato.
0	0	1	Shift sotto il controllo del Timer 2.
0	1	0	Shift sotto il controllo degli impulsi di $\phi_2$ .
0	1	1	Shift sotto il controllo degli impulsi di un clock esterno.
1	0	0	Uscita free-running alla frequenza determinata dal Timer 2.
1	0	1	Shift out sotto il controllo del Timer 2.
1	1	0	Shift out sotto il controllo degli impulsi di $\phi_2$ .
1	1	1	Shift out sotto il controllo degli impulsi del clock esterno.

Figura 2-33: Controllo del Registro di Shift

*In uscita* l'utilizzatore caricherà il registro di shift. Ciò farà iniziare automaticamente la temporizzazione e il processo di shift. Quando 8 bit sono stati trasportati fuori dal registro, si posizionerà automaticamente il flag di interrupt (bit 2 del registro del flag di interrupt). Questo può essere così rilevato dal programma.

*In ingresso*, per ottenere l'inizio del processo di temporizzazione, il registro di shift deve essere inizializzato con qualche valore, ad esempio "0". In seguito a ciò il processo di caricamento dei bit inizierà e procederà alla frequenza fissata dalla temporizzazione, ad esempio da timer 2, la fase 2 del clock oppure un clock esterno, a seconda di ciò che è stato impostato sui bit 2, 3, 4 dell'ACR. Quando si sono accumulati 8 bit sarà posizionato il corrispondente flag di interrupt di IFR. Il programma metterà un valore, ad esempio "0", nell'SR, poi sonderà continuamente il livello del bit 2 di IFR. Quando si rileva un interrupt lo shift è completo. Il registro di shift sarà quindi disabilitato azzerando i bit 2, 3, 4 di ACR, mentre il programma memorizza i dati. Naturalmente se i dati entrano in continuazione, il registro di shift non sarà disabilitato e il programma "tornerà indietro" abbastanza velocemente per non perdere i dati.

## PROGRAMMAZIONE DEL 6522

Il 6522 è una combinazione di PIO, timer e shifter. Le operazioni di base sul PIO si eseguono

come sul 6520, ad eccezione del fatto che i registri possono essere selezionati direttamente e che non necessita posizionare il bit 2 del registro di controllo per diversificarli. Ad ogni modo le possibilità di controllo del 6522 sono vaste e abbastanza diverse da quelle del 6520. Esaminiamo innanzi tutto quindi, alcuni esempi base di ingresso-uscita, poi alcuni esempi di opzioni di controllo.

### Operazioni di Base per l'Ingresso

La predisposizione ad ingresso la si ottiene caricando tutti zeri nel registro di direzione della porta che deve funzionare come ingresso, poi leggendo il contenuto dell'ORB. Nel seguente semplice programma, si memorizzano inoltre, i dati appena letti, nella locazione 20 di memoria. Il programma è il seguente:

```

INPUT      LDA      #0
           STA      DDRA      LA PORTA A È L'INGRESSO
           LDA      ORA      LEGGI I DATI (SE PRESENTI)
           STA      $20      METTI I DATI IN MEMORIA
  
```

RS3	RS2	RS1	RS0	R/W	REGISTRO	COMMENTO
0	0	0	0	W	ORB	controlla handshake
0	0	0	0	R	IRB	
0	0	0	1	W	ORA	
0	0	0	1	R	IRA	
0	0	1	0	-	DDRB	
0	0	1	1	-	DDRA	
8	1	8	8	W	T1L-L	latch contatore T1L-L in T1C-L
0	1	0	1	R	T1C-L	
0	1	0	1		T1C-H	
0	1	1	0		T1L-L	latch contatore trigger T2L-L in T2C-L
0	1	1	1		T1L-H	
1	0	0	0	W	T2L-L	
1	0	0	0	R	T2C-L	
1	0	0	1		T2C-H	
1	0	1	0		SR	
1	0	1	1		ACR	
1	1	0	0		PCR	
1	1	0	1		IFR	su handshake nessun effetto
1	1	1	0		IER	
1	1	1	1		ORA	

Figura 2-34: La Selezione Registro nel 6522 è Diretta

### Operazioni di Base per l'Uscita

La predisposizione ad uscita si ottiene esattamente come per l'ingresso; il registro di direzione dati è caricato con tutti uni, specificando così tutte uscite. Supponiamo che i dati da inviare alla porta B siano nella locazione 20 di memoria, in modo che debbano essere prima caricati nell'accumulatore poi trasferiti all'ORB. È necessaria quindi un'istruzione extra per trasferire prima i dati nell'accumulatore, poi dall'accumulatore all'ORB. Il programma è il seguente:

```

OUTPUT    LDA      #$FF      B = USCITA
           STA      DDRB
           LDA      $20      PRENDI I DATI DALLA MEMORIA
           STA      ORB      MANDALI IN USCITA
  
```

## Uso delle Opzioni di Controllo

Si vuole qui, far assumere alla porta A la configurazione di tutti ingressi. Si supponrà che la periferica o il dispositivo collegato alla porta A invii lo strobe di "dati pronti" sulla linea CA1. Lo strobe sarà attivo durante la sua transizione basso-alto. Il 6522 rileverà questa transizione di strobe "dati pronti" ed il programma controllerà il 6522 per vedere se sono stati ricevuti dei dati. Se sono stati ricevuti li leggerà e li memorizzerà nella locazione 20 di memoria. Il programma è già stato sviluppato (vedi "Operazioni di Base per l'Ingresso" pagina 28), ed è riportato di seguito:

```
READYIN  LDA    #0      A = INGRESSO
          STA    DDRA
          LDA    #1      CA1 INT LO AD HI
          STA    PCR
TEST      LDA    IFR      TEST DEL BIT 1
          AND    #$2      00000010 BINARIO
          BEQ    TEST     UGUALE A 1?
          LDA    ORA      LEGGI I DATI
          STA    $20      METTILI IN MEMORIA
```

Di solito per predisporre ORA come tutti ingressi, si caricano tutti zero nel registro di direzione dati:

```
LDA    #0
STA    DDRA
```

Deve ora essere condizionato il registro di controllo PCR, in modo che si generi interrupt quando avviene una transizione basso-alto:

```
LDA    #1
STA    PCR
```

Le due istruzioni sopra riportate caricano il valore binario 00000001 nel PCR. Facendo riferimento alla Figura 2-23, il lettore può verificare che questo è davvero il valore corretto. Il bit zero del registro di controllo periferico PCR, indica che bisogna rilevare una transizione attiva del segnale d'ingresso. Dal momento che noi vogliamo che il flag di interrupt di CA1 sia posizionato per una transizione positiva (basso-alto), bisogna mettere ad 1 il PCR0.

I bit 6 e 7 del ACR sono relativi al modo di funzionamento del timer 1. Poiché non stiamo usando il timer, il loro contenuto qui, non ha alcuna importanza. I bit 2, 3 e 4 del ACR indicano le operazioni del registro di shift. Dal momento che qui il registro di shift non è usato, sarà zero, come indicato in Figura 2-33. Il bit 5 del ACR è il controllo di T2 quindi, in questo caso non è usato. Il bit 1 è l'abilitazione del latch PB, anch'esso non è usato in questo caso. Il bit zero è l'abilitazione del latch della porta A. Quando, indicati (scrivendo un "1") i dati presenti sull'ingresso A subiranno un latch, quando il flag di interrupt di CA1 è uguale ad "1". Ciò sarà eseguito, con:

```
LDA    #1
STA    ACR
```

Poiché noi qui supponiamo sia usato un polling invece di un interrupt hardware, sarà responsabile il programma della lettura del contenuto del flag di interrupt e del controllo della presenza di interrupt. Il contenuto del registro del flag di interrupt è mostrato in Figura 2-28. Il bit di posizione 1 del IFR, deve essere controllato per sapere se è stato ricevuto il segnale CA1 di "dati pronti". Questo lo si ottiene con le seguenti tre istruzioni:

```
TEST     LDA    IFR
          AND    #$2
          BEQ    TEST
```

L'istruzione AND elimina tutti i bit tranne il bit di posizione 1, di modo che, esso può essere controllato.

Finchè il bit 1 è zero questo programma rimarrà nel ciclo di polling. Non appena si rileva il segnale di "dati pronti" questi possono essere letti dal ORA e trasferiti nella loro locazione di memoria finale, che supponiamo essere, come al solito, la locazione 20 di memoria:

```
LDA    ORA
STA    $20
```

Trasferendo il contenuto del ORA nell'accumulatore, si cancellerà automaticamente il bit 1 di IFR (indicatore di stato di CA1), di modo che l'interrupt sarà automaticamente azzerato.

È importante ricordare che i flag di interrupt devono essere cancellati ogni volta che vengono usati. Il 6522 è organizzato in modo che le operazioni "normali", come la lettura del contenuto di ORA dopo il rilevamento di un interrupt, faranno quanto sopra automaticamente. Ad ogni modo avvisiamo il lettore del fatto che se egli userà "programmazione non standard", potrà commettere l'errore di lasciare continuamente ad "1" i flag di interrupt. Una tecnica che può essere usata in questo caso è riscrivere il contenuto di IFR dopo averlo letto:

```
STA    IFR
```

Questo "trucco nella programmazione" azzererà solamente il bit che è appena stato posto al valore "1", senza modificare gli altri (a meno che un bit fosse "1").

### Un Protocollo di Handshake sull'Ingresso

Supporremo che venga usata la sequenza completa di handshake: innanzi tutto il programma è responsabile dell'invio di un impulso di "start" (attivo alto) al dispositivo. Più tardi il dispositivo risponderà con uno strobe di "dati pronti" (qui attivo da alto-basso) e il programma sarà responsabile del controllo dell'avvenuta ricezione del segnale, poi trasferirà il dato nella locazione 20 di memoria. Il programma è il seguente:

```
NSHAKE  LDA    #0
        STA    DDRA      A È UN INGRESSO
        STA    ACR
        LDA    #$0C      BIT 2 E 3 ALTI
        STA    PCR       AZZERA L'IMPULSO DI START
        LDA    #$0E      BIT 1, 2, 3 ALTI
        STA    PCR       GENERA START SU CA2
        LDA    #$0C
        STA    PCR
WAIT     LDA    IFR       AZZERALO
        AND    #$02      INTERRUPT?
        BEQ    WAIT      (IMPULSO DI START?)
        LDA    ORA       CICLO DI POLLING
        STA    $20       DATI PRONTI
        STA    $20       METTILI IN MEMORIA
```

Esaminiamo il programma. Come al solito la porta A è programmata come ingresso memorizzando zero nel DDRA:

```
LDA    #0
STA    DDRA      ZERO IN DDRA
STA    ACR
```

Supponiamo che non sia necessario latching in ingresso (vedere i precedenti programmi se si vuole latch in ingresso). Il registro PCR ora deve essere caricato in modo che sia generato un impulso di start attivo alto. Il livello di CA2 (la linea che useremo per fornire il segnale di

start CA1 può essere usata solo come ingresso) sarà prima messo basso, poi alto per garantire una transizione basso-alto. Per portare l'uscita CA2 al livello basso si carica il valore "110" rispettivamente nei bit 3, 2 e 1 del PCR (vedi Figura 2-24). Ciò si effettua con le seguenti istruzioni:

```
LDA    #$0C    00001100
STA    PCR
```

Successivamente il valore su CA2 deve essere posto ad "1". Ciò si ottiene caricando il valore "111" nei bit 3, 2, 1 del PCR:

```
LDA    #$0E    00001110
STA    PCR
```

Supporremo che sia sufficiente un breve impulso per fornire il segnale di "start". Alcuni dispositivi possono richiedere un impulso di durata maggiore. In tal caso occorrerà aggiungere un ritardo in questo punto, per garantire che l'impulso rimanga alto durante il periodo di tempo stabilito. Qui noi riporteremo semplicemente il segnale al livello basso:

```
LDA    #0C      00001100
STA    PCR
```

Ora si procede come nel programma precedente, con il polling del bit uno del IFR, per controllare se CA1 è stato posto ad uno:

```
WAIT    LDA    IFR    00000010
        AND    #$02
        BEQ    WAIT
```

In seguito, come sopra, il dato è letto da ORA e memorizzato nella locazione 20 di memoria:

```
LDA    ORA
STA    $20
```

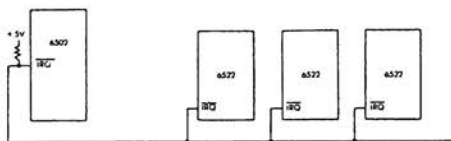


Figura 2-35: Collegamento Multiplo di 6522 — Generazione di un IRQ

## Impiego Multiplo di 6522

Nel caso in cui si impieghino diversi 6522, la loro uscita IRQ per richiesta di interrupt è di solito, collegata alla linea di IRQ, come illustrato in Figura 2-35. In ogni caso, quando il 6502, riceverà un IRQ, il programma dovrà determinare quale 6522 lo ha generato. In genere è usato un ciclo di polling. In tale ciclo si interrogheranno a turno gli IFR di ciascun dispositivo per determinare quale abbia richiesto interrupt. Questa informazione è certamente disponibile sul bit

7 del registro dei flag di interrupt, come si può vedere in Figura 2-22. Il lettore ricorderà che il bit 7 è universalmente usato come posizione preferenziale per polling, in quanto il contenuto del registro sotto controllo è caricato nell'accumulatore e il valore del bit 7 condiziona il bit del segno del microprocessore nel registro dei flag (bit N). Con l'istruzione successiva, poi, il programma può facilmente controllare il bit N e vedere se era "0" o "1". Questo è esattamente ciò che il programma di polling fa qui. Un programma di polling tipico è riportato di seguito:

```

                LDA     IFR1
                BPL     NEXT1
INTFOUND1...   (IDENTIFICAZIONE DI 1 DELLE 7 CAUSE)
                .
                .
                .
NEXT1          LDA     IFR2
                BPL     NEXT2

```

Il programma carica il contenuto del IFR del primo 6522 e controlla se è positivo. In caso affermativo, non è stato generato interrupt dal dispositivo e il programma controlla il successivo e così via. Nel caso in cui si trovi il dispositivo che ha generato l'interrupt, una routine specifica deve poi determinare cosa fare. Esaminiamo tale routine.

### *Identificazione di uno dei 7 Possibili Interrupt Interni del 6522*

Con riferimento alla Figura 2-22, si nota che sette possibili condizioni possono generare un interrupt nel registro IFR del 6522: T1, T2, CB1, CB2, SR, CA1, CA2. Se sono usate contemporaneamente tutte le risorse interne del 6522, come spesso succede, si dovranno poi valutare tutte le possibilità. Di seguito è riportato un programma che individua uno dei 7 interrupt:

```

ONEOF7        ASL     A
                BMI    TIMER 1
                ASL     A
                BMI    TIMER 2
                ASL     A
                ...

```

Il programma controlla in successione i bit 6, 5, 4, etc. con un semplice shift a sinistra, di un bit alla volta, del contenuto dell'accumulatore. Si noti che l'ordine in cui viene eseguito lo shift stabilisce una priorità degli interrupt all'interno del dispositivo. Impiegando il programma illustrato sopra, il timer 1 avrà la più alta priorità, poi viene il timer 2 etc. Si può volere assegnare una priorità diversa agli interrupt, controllando i bit in un ordine diverso.

### *Generazione di Ritardi con un timer*

Il lettore potrà studiare i particolari dei timer nei data sheet dei costruttori, prima di usarli. Il timer 2 è più semplice del timer 1. I timer non sono uguali ed è importante capire le loro caratteristiche specifiche prima di usarli. Poiché uno studio completo dei modi di funzionamento dei timer non è necessario per gli scopi di questo libro, mostreremo due esempi tipici di generazione di ritardi, usando rispettivamente il timer 2 e il timer 1. Altri esempi saranno illustrati nei capitoli delle applicazioni.

## Generazione di un Ritardo One-Shot del Timer 2

Il programma è il seguente:

```
ONESHOT2  LDA      #0
           STA      ACR          SCELTA DEL MODO
           STA      T2LL        LOW – LATCH = 0
           LDA      #$01        DURATA DEL RITARDO
           STA      T2CH        PARTE ALTA = 01 ESADEC. START
           LDA      #$20        MASCHERA
LOOP       BIT      IFR          FINE TEMPO?
           BEQ      LOOP
           LDA      T2CL        AZZERAZIONE DELL'INTERRUPT DEL TIMER 2
```

I bit 6 e 7 di ACR devono essere posti al valore 0 per indicare il modo one-shot (PB7 non è usato con T2). Dal momento che supponiamo di non usare nessuna delle altre risorse, come ad esempio il registro di shift, carichiamo semplicemente tutti zeri nel registro ACR:

```
LDA      #0
STA      ACR
```

Il timer 2, come il timer 1, contiene un OR a 16 bit, cosicché le due parti del registro devono essere caricate separatamente. Caricheremo prima la parte più bassa poi quella più alta:

```
STA      T2LL
LDA      #$01
STA      T2CH
```

Caricare il valore \$01 nel T2C-H, significa anche azzerare il flag di interrupt e far partire automaticamente il contatore.

La Figura 2-28 mostra che il bit 5 del IFR indica la fine corsa del timer 2. Bisogna quindi controllare se il bit 5 del IFR è al valore "1". Questo si effettua con le seguenti tre istruzioni:

```
      LDA      #$20      BIT 5 = 1
LOOP  BIT      IFR
      BEQ      LOOP
```

Il valore 20 esadecimale, è uguale a "00100000". Tale valore lo si usa per controllare se il bit 5 è uguale a "1". L'istruzione BIT esegue un AND logico senza modificare il contenuto dell'accumulatore. Finché il bit 5 rimane a "0", il programma resta in loop in attesa dell'interrupt del timer 2. Quando il timer 2 genera interrupt, esso è rilevato, e il programma esce dal loop.

Infine il programma deve azzerare esplicitamente l'interrupt del timer 2 prima di svolgere qualsiasi altro compito. Questo può essere ottenuto caricando un nuovo valore nel registro del contatore. Ad ogni modo, dal momento che questo programma può essere usato in qualunque caso, non facciamo alcuna ipotesi su quello che sarà fatto in seguito. Il flag di interrupt sarà azzerato, sia scrivendo in T2C-H che leggendo in T2C-L. Dal momento che non si vuole far ripartire il contatore, non scriveremo in T2C-H, ma leggeremo invece T2C-L, per azzerare semplicemente l'interrupt:

```
LDA      T2CL
```

## Generazione di un Ritardo One-Shot del timer 1

Qui useremo il timer 1 in modo sostanzialmente analogo al timer 2 di cui sopra. Il timer 1, ad

ogni modo, a differenza del timer 2, è munito di un registro di latch a 16 bit. Il programma è il seguente:

```

ONESHOT1  LDA    #0
           STA    ACR      MODO 1-SHOT-NESSUN IMPULSO SU PB7

           STA    T1LL     LOW LATCH
           LDA    #$01     RITARDO
           STA    T1CH     CARICA ANCHE T1CL E PARTI

           LDA    #$20
LOOP       BIT    IFR      FINE TEMPO?
           BEQ    LOOP
           LDA    T1LL     AZZERA IL FLAG DI INTERRUPT

```

Il programma è sostanzialmente analogo al precedente e si spiega da solo. L'unica differenza è che prima si carica la parte bassa del latch, poi il programma scrive in T1C-H la parte alta del contatore opportuno. Questa istruzione consiste anche nel trasferimento del contenuto di T1L-L in T1C-L (vedi Figura 2-34 che mostra i registri interni del 6522) e nella partenza del contatore. Il resto del programma è uguale.

### *Generazione di un Impulso*

I programmi precedenti generano un ritardo per un programma. Se si deve generare un impulso, bisogna allora indicare l'opportuno pin d'uscita. Per il timer 1 si impiegherà il pin PB7 per fornire l'impulso d'uscita; PB7 sarà un'uscita se DDRB7 o ACR7 sono uguali a "1".

Il timer 2 non invia direttamente un impulso su un pin d'uscita. L'impulso deve essere generato aggiungendo istruzioni che pongano ad "1" od a "0" esplicitamente uno dei bit della porta. Ad ogni modo il timer 2 può facilmente contare impulsi nel suo modo pulse-counting. Il pin PB6 è usato a questo scopo. Ciò sottolinea di nuovo le differenze pratiche fra questi timer. Il lettore è incoraggiato ad esaminare i data sheet del costruttore in ogni applicazione pratica, per trarne il massimo vantaggio.

### *Shifting in Ingresso e Uscita*

Il registro di shift SR è collegato al pin CB2 del 6522. Tutti gli impulsi saranno generati o rilevati su questo specifico pin. La combinazione dei bit 2, 3, e 4 di ACR determina il modo di funzionamento dello shifter. Le otto combinazioni, sono illustrate nella precedente Figura 2-33.

Negli esempi fin qui fatti il contenuto dei bit 2, 3, e 4 di ACR è sempre stato zero, in modo che il registro shifter era disabilitato. Tale registro compirà shift in ingresso o in uscita sotto il controllo di una delle tre possibili sorgenti di temporizzazione: timer 2, fase 2 del clock oppure clock esterno. Inoltre, esso fornisce un altro modo speciale con un'uscita in free-running al ritmo determinato dal timer 2. Il lettore è nuovamente rimandato ai data sheet del costruttore per le specifiche complete dello shifter. Qui saranno presentati solamente due esempi tipici di shifting in ingresso e in uscita.

### *Shifting con un Clock Esterno*

Il programma è il seguente:

```

SHIFTIN   LDA    #0
           STA    ACR      AZZERA SR
           LDA    #$0C     MODO CLOCK ESTERNO
           STA    ACR      START SHIFTER
LOOP       LDA    IFR      C'È FLAG?
           AND    #$04     TEST BIT 2

```



BEQ	LOOP	LOOP D'ATTESA
LDA	SR	METTI GLI 8 BIT NELL'ACC
STA	\$20	METTI IN MEMORIA

Il registro di shift è inizialmente azzerato caricando zero in ACR:

LDA	#0
STA	ACR

Quindi è specificato il modo di funzionamento corretto, caricando il valore "011" rispettivamente nei bit 4, 3, 2, di ACR:

LDA	#\$0C
STA	ACR

Questo indica uno shift in ingresso sotto il controllo di un clock esterno (vedi Figura 2-33).

Una volta che lo shift è avvenuto, il meccanismo di shifting è automaticamente disabilitato, ed il flag di interrupt di SR è posizionato nel registro IFR. Dopo che lo shift ha avuto inizio, quindi, il programma controlla semplicemente il contenuto del bit di posizione 2 del IFR (vedi Figura 2-28) per verificare se è "1". Il ciclo di polling è il seguente:

LOOP	LDA	IFR
	AND	#\$04
	BEQ	LOOP

A questo punto il contenuto del registro di shift SR deve semplicemente essere trasferito nella locazione 20 di memoria come al solito:

LDA	SR
STA	\$20

### *Shifting in Uscita sotto il Controllo della Fase 2*

Il programma è sostanzialmente simile a quello sopra, a meno dei bit di controllo, che devono essere caricati in ACR, per indicare adeguatamente il modo di funzionamento. Supponendo che si debba trasferire fuori semplicemente una parola di 8 bit, non è necessario un ciclo di attesa per determinare se lo shift è finito oppure no. Il programma è il seguente:

SHIFTOUT	LDA	#0	
	STA	ACR	AZZERA SR
	LDA	#\$18	
	STA	ACR	MODO D'USCITA 2
	LDA	\$20	LEGGI IL DATO DALLA MEMORIA
	STA	SR	

Come sopra, il registro di shift è prima azzerato, poi si carica ACR con il valore "18" esadecimale, che indica la combinazione "110" nelle posizioni dei bit 4, 3 e 2. Questo specifica che lo shift avviene in uscita a un ritmo controllato dalla fase 2 del clock di sistema:

LDA	#0
STA	ACR
LDA	#\$18
STA	ACR

I dati sono poi estratti dalla locazione di memoria 20, e depositati nel registro di shift. Inserendo i dati nel registro esso automaticamente parte.

LDA	\$20
STA	SR

Se si deve inviare una successione di parole a 8 bit, tale programma aspetterà la fine di uno shift prima di iniziare il successivo. Ciò sarà compiuto tramite un ciclo di attesa come nel caso precedente. Una volta che gli 8 bit sono stati inviati fuori, il 6522 pone automaticamente ad "1" il bit 2 di IFR (vedi Figura 2-28). Il programma quindi dovrà semplicemente controllare in continuazione il bit 2 di IFR, fino a che non assume il valore "1". Non appena è stato rilevato il valore "1" lo shift ricomincerà.

## Sommario del 6522

Le tre funzioni di questo componente sono: PIO, timer, shift. Inoltre possono essere indicati segnali di controllo complessi per il PIO e il timer. Si sono descritte le funzioni dei possibili segnali di controllo e le opzioni. Tale componente può essere visto come un insieme di tre funzioni separate. Le funzioni della porta A e della porta B sono sostanzialmente simili ma non simmetriche: i due timer hanno alcune caratteristiche comuni ma offrono possibilità diverse. Infine, il registro di shift è essenzialmente simmetrico in ingresso e in uscita e può essere usato per ricevere o trasmettere bit oppure parole ad ogni frequenza, da un certo numero di sorgenti di clock esterne.

**ESERCIZIO 2-4:** *Mettere in una tabella di memoria a 2 parole posta alla locazione BUFFER, due successive parole di dati da DEVICE 1. DEVICE 1 fornisce uno strobe READY, attivo basso-alto. È richiesto un segnale di accettazione (impulso alto).*

**ESERCIZIO 2-5:** *Lo stesso dell'esercizio 2-4, a differenza del fatto che DEVICE 1 richiede un impulso di START attivo basso e risponde con il segnale READY.*

**ESERCIZIO 2-6:** *Inviare dati a DEVICE 2 dalla locazione di memoria BUFFER. DEVICE 2 invia un segnale BUSY quando non è pronto.*

**ESERCIZIO 2-7:** *Come l'esercizio 2-5, ma DEVICE 2 richiede uno strobe STATUS per fornire una risposta READY/BUSY.*

**ESERCIZIO 2-8:** *Accendere una stampante con un «1» sulla linea di controllo. attendere il READY, inviare un carattere e spegnerla.*

**ESERCIZIO 2-9:** *Contare 10 impulsi in ingresso su PB6.*

**ESERCIZIO 2-10:** *Generare un impulso di 1 ms su PB7.*

**ESERCIZIO 2-11:** *Eseguire uno shift in uscita di 8 bit dalla locazione di memoria BUFFER alla velocità del timer 2.*

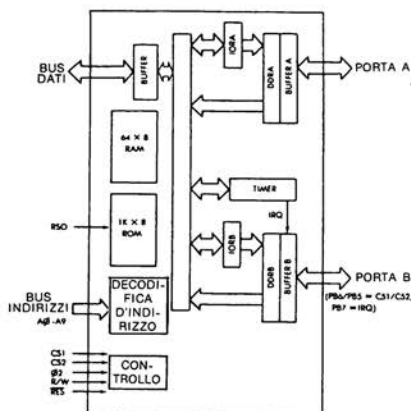


Figura 2-36: Architettura Interna del 6530

## IL 6530 ROM-RAM I/O TIMER (RRIOT) (RRIOT sta per ROM-RAM-I/O-Timer)

Il 6530 è una combinazione speciale di componenti che riunisce quattro funzioni abitualmente distinte: un PIO, un timer, una RAM e un ROM. L'architettura interna del 6530 è illustrata in Figura 2-36. Esso è costituito dal solito PIO a due porte, ognuna delle quali con il proprio registro di direzione dati. Ad ogni modo non c'è nessuna linea di controllo o interrupt logico associato alle porte. Il timer è collegato alla porta B. La memoria RAM fornisce 64 byte, la ROM 1 K byte. Una ROM, una volta programmata, non può essere cambiata. Dal momento che non è economico produrre ROM in piccole quantità, il 6530 è usato solamente in situazioni dove si deve produrre un grande numero di componenti identici. Ad esempio, la scheda KIM, usa due 6530 che contengono il programma di controllo Interno o "monitor".

In questo componente tre pin hanno doppia funzione: CS1 e CS2 sono opzioni di maschera invece di PB6 e PB5. Inoltre, PB7 può essere usato come richiesta di interrupt IRQ.

## Il Timer d'Intervallo

Il timer di intervallo è fornito di un registro ad 8 bit e può essere usato in uno dei 4 possibili modi. In funzione dei valori A0 e A1 delle linee d'indirizzo, esso conterà, con incrementi di 1, 8, 64, 1024 volte, il clock del sistema. Al programmatore il timer appare come l'insieme di 4 locazioni di memoria, come illustrato in Figura 2-37.

Quando si usa il timer, il pin PB7, può essere usato come un pin d'interrupt. Quando è usato come interrupt, il PB7 deve essere programmato come ingresso. Quando non è usato come interrupt, può essere usato per qualsiasi scopo. Per i dettagli sull'utilizzazione di PB7 come interrupt, il lettore è rimandato ai data sheet del costruttore.

A2	A1	A0		
0	0	0	BUFFER A	
0	0	1	DDRA	
0	1	0	BUFFER B	
0	1	1	DDRB	
1	0	0	TIMER 1T	* IRQ a PB7
1	0	1	(W) TIMER 8T (R) INT FLAG	NO IRQ a PB7
1	1	0	TIMER 64T	* IRQ a PB7
1	1	1	TIMER 1024T (R) INT FLAG	NO IRQ a OB7

Nota: A3 indica se è usato interrupt.

Figura 2-37: Mappa di Memoria del 6530

## IL RIOT 6532

Il 6532 è sostanzialmente un 6530 senza la ROM. La RAM comunque, è più grande: essa fornisce 128 parole. Inoltre la linea PA7, su tale dispositivo, può essere usata come un ingresso rilevatore di fronti. Quando questo modo è usato in corrispondenza di una transizione attiva, si posiziona un flag interno di interrupt (bit 6 del registro flag di interrupt).

L'architettura interna del 6532, è illustrata in Figura 2-38. L'indirizzamento del chip è mostrato in Figura 2-39. Il resto delle operazioni del 6532 sono sostanzialmente quelle del 6530.

Le porte A e B non sono simmetriche. La maggior differenza fra le due porte è che la porta B è fornita di buffer in push-pull che sono capaci di fornire 3mA ad 1,5 volt.

Ciò permette il collegamento diretto di questa porta a LED o a transistori Darlington. Inoltre la porta A legge direttamente dai pin. Sulla porta B, i dati sono letti dal registro d'uscita invece che dai pin periferici.

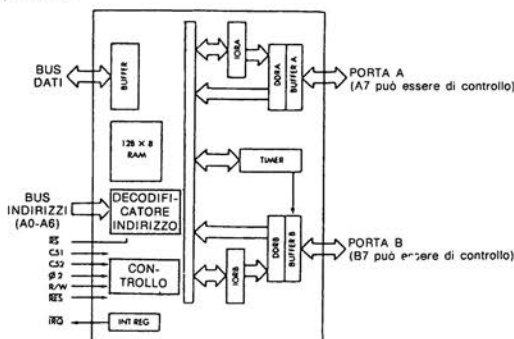


Figura 2-38: Architettura Interna del 6532

RS	A4	A3	A2	A1	A0	R/W	SELEZIONE
0	-	-	-	-	-	-	RAM
1	-	-	0	0	0	-	ORA
1	-	-	0	0	1	-	DDRA
1	-	-	0	1	0	-	ORB
1	-	-	0	1	1	-	DDRB
1	1	*	1	0	0	0	SCRITTURA TIMER-11
1	1	*	1	0	1	0	+8T
1	1	*	1	1	0	0	+64T
1	1	*	1	1	1	0	+1024T
1	-	-	1	-	0	1	LETTURA TIMER
1	-	-	1	-	1	1	LETTURA FLAG INTERRUPT
1	0	-	1	**	***	0	SCRITTURA CONTROLLO FRONTE RILEVATO

\* disabilita (0) / abilita (1) INT da timer a IRQ

\*\* disabilita (0) / abilita (1) INT da PA7 a IRQ

\*\*\* negativo (0) / positivo (1) fronte rilevato

Figura 2-39: Indirizzamento del 6532

## SOMMARIO

La maggior parte delle applicazioni richiederà almeno l'uso di due o più porte su uno o più PIO e l'impiego di un timer programmabile. Applicazioni sofisticate richiederanno l'uso di segnali di controllo e il possibile impiego di shift automatizzati. Tutti i componenti analizzati — il 6520, 6522, 6530 e 6532 — forniscono due porte PIO. Ad eccezione del 6520, essi forniscono tutti almeno un timer programmabile. In Figura 2-40 è riportata una tabella di confronto dei quattro dispositivi ingresso-uscita.

Uno o più di uno dei suddetti PIO saranno in tutte le applicazioni di questo libro.

	6520	6522	6530	6532
LINEE PORTA A	8	8	8	8
LINEE PORTA B	8	8	5 a 8	8
LINEE DI CONTROLLO A	2	2	0	0
LINEE DI CONTROLLO B	2	2	0	0
DDRA	1	1	sì	sì
DDRB	1	1	sì	sì
TIMER 1	-	sì	sì	sì
TIMER 2	-	sì	-	-
ROM	-	-	1K x 8	-
RAM	-	-	64 x 8	128 x 8
ALTRO	-	somma i registri di controllo	rapporti 4 timer opzionale	rapporti 4 timer
INTERRUPT	2	1		1

Figura 2-40: Tabella di Confronto dei Quattro PIO



## CAPITOLO 3

# SISTEMI BASATI SUL 6502

### INTRODUZIONE

Le applicazioni presentate in questo volume faranno riferimento a un sistema 6502 "standard". Innanzi tutto sarà, quindi, presentata l'organizzazione di un tale "sistema standard". Saranno poi descritte alcune linee reali 6502, che mostreremo essere coerenti con il modello standard appena introdotto.

Con lo scopo di presentare applicazioni reali, è necessario definire una precisa configurazione hardware, a cui queste fanno effettivamente riferimento. La maggior parte degli esempi presentati nel libro, sono direttamente applicabili alla scheda SYM e possono essere facilmente adattati alla scheda KIM. Un paragrafo del capitolo successivo presenterà in particolare, programmi KIM. Non si sostiene nessuna scheda e nessun costruttore. Semplicemente, a scopo educativo, è più pratico presentare applicazioni direttamente realizzabili a schede esistenti, invece di inventarne una fittizia. Molti programmi scritti per il SYM sono compatibili con il KIM e possono essere facilmente adattati ad altre schede, ad esempio l'AIM65. Il lettore è sollecitato ad esercitarsi nel decidere quale scheda soddisferà, in modo migliore, alle sue necessità.

In questo capitolo sono presentate le architetture del KIM, SYM e AIM65. Il SYM è presentato in maggior dettaglio, cosicché il lettore che non ha un SYM, possa comprendere le interconnessioni usate nei programmi delle applicazioni, presentate nei successivi capitoli. Ad ogni modo sottolineiamo di nuovo che può essere usata ogni altra scheda e che i cambiamenti richiesti nei programmi sono in genere minimi.

### UN SISTEMA 6502 "STANDARD"

Ogni sistema standard a microprocessore, comprende almeno il microprocessore (MPU), il suo circuito di clock, la ROM, la RAM e uno o più PIO. In Figura 3-1, è mostrata l'organizzazione di un tale sistema standard che fa uso del 6502.

Il 6502 contiene la maggior parte dei circuiti di clock all'interno del chip del microprocessore, cosicché è necessario solamente un quarzo esterno ad un circuito oscillante. Il 6502 e il suo circuito di clock, sono mostrati sulla sinistra dell'illustrazione. Il 6502, come ogni microprocessore "standard", fa uso di tre bus: bus degli indirizzi (16 linee), il bus dei dati (8 linee bidirezionali) e infine il bus di controllo.

Nel sistema standard la memoria RAM (memoria di lettura-scrittura), la memoria ROM (memoria di sola lettura) e il PIO sono indicati come chip separati collegati ai 3 bus. La ROM tipicamente conterrà un programma *monitor* necessario per sfruttare le risorse del microprocessore, o anche il programma dell'utente (in applicazioni industriali). Il PIO mette a disposizione due porte (di 8 linee ciascuna) per le comunicazioni con dispositivi esterni, più, a volte, alcune linee di controllo. In ogni applicazione pratica, saranno necessari almeno due PIO per fornire un numero sufficiente di linee I/O. In genere è necessaria una po' di logica per la decodifica degli indirizzi e per altre funzioni.

Poiché nella famiglia 6502 sono disponibili parecchie combinazioni di chip, la ROM, la RAM, il PIO, possono essere posti su uno o più chip.

Ad ogni modo ogni sistema che fa uso del 6502, normalmente, comprenderà tutti gli elementi logici di Figura 3-1.

Esaminiamo ora alcuni esempi reali e come sono collegati alla nostra linea standard.

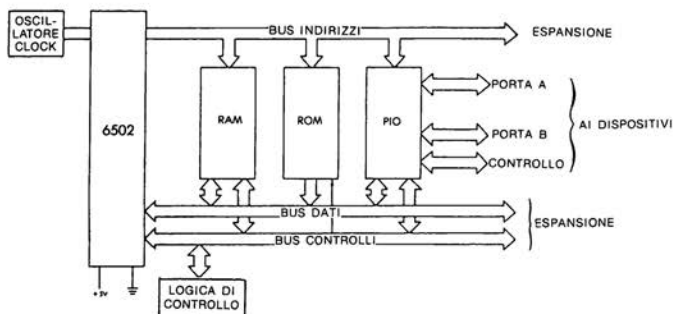


Figura 3-1: Organizzazione di un Sistema 6502 "Standard"

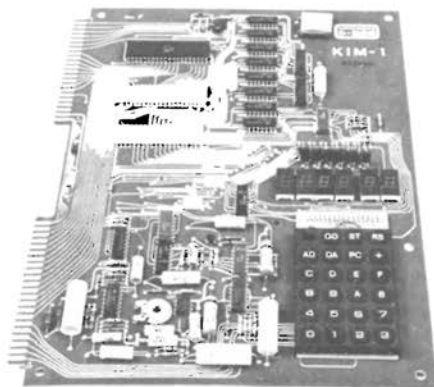


Figura 3-2: Fotografia del KIM-1



## IL KIM-1

Il KIM-1 fu una prima scheda introdotta dalla MOS Technology, in supporto del suo microprocessore 6502. Essa incorpora un numero minimo di componenti, è fornita di una tastiera esadecimale e di 6 LED, in modo che può essere usata come un microcomputer completo standard a basso costo. Essa è mostrata in Figura 3-2. La sua organizzazione interna è illustrata in Figura 3-3.

Il KIM-1 comprende una RAM di 1 K per 8 separata, (per l'utente) e due chip di combinazioni 6530. Il lettore ricorderà, dal capitolo precedente, che il 6530 è un chip combinazione che fornisce un PIO, un timer programmabile, una ROM ed una RAM. Su tale sistema non c'è bisogno di una memoria ROM esterna, dal momento che la quantità di ROM fornita dai due 6530, è sufficiente a contenere il monitor del sistema. Ogni 6530 contiene anche 64 bytes di RAM, che sono in parte usati dal monitor del sistema.

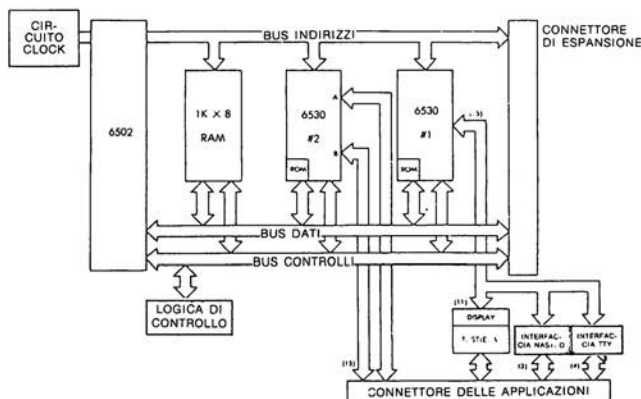


Figura 3-3: Organizzazione Interna del KIM-1

Il sistema è munito inoltre, di una tastiera, 6 LED, un'interfaccia per registratore su nastro e un'interfaccia per stampante. È possibile espandere esternamente il complesso per mezzo di due connettori, chiamati rispettivamente connettore di espansione e connettore di applicazione, come illustrato in Figura 3-3. In Figura 3-4 è mostrata la mappa di memoria del sistema. I segnali per i due connettori del KIM sono mostrati nelle Figure 3-5 e 3-6.

Il lettore può controllare che l'organizzazione di questo sistema è in accordo con la descrizione del nostro sistema standard 6502 mostrato in Figura 3-1. I dettagli sui pin connessi sono utili a quei lettori che vorranno collegare le applicazioni qui presentate a questa scheda particolare.

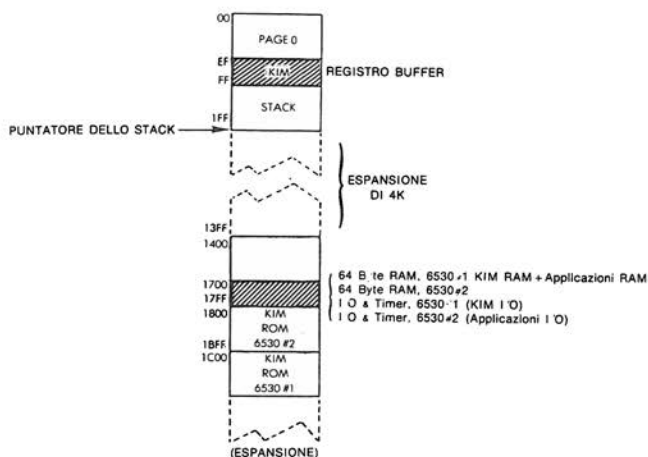


Figura 3-4: Mappa di Memoria del KIM-1

22	KB Col D	Z	KB Riga 1
21	KB Col A	Y	KB Col C
20	KB Col E	X	KB Riga 2
19	KB Col B	W	KB Col G
18	KB Col F	V	KB Riga 3
17	KB Riga 0	U	TTY PTR
16	PB5	T	TTY KYBD
15	PB7	S	TTY PTR RTRN (+)
14	PA0	R	TTY KYBD RTRN (+)
13	PB4	P	AUDIO OUT HI
12	PB3	N	+ 12 V
11	PB2	M	AUDIO OUT LO
10	PB1	L	AUDIO IN
9	PB0	K	DECODE ENAB
8	PA7	J	K7
7	PA6	H	K5
6	PA5	F	K4
5	PA4	E	K3
4	PA1	D	K2
3	PA2	C	K1
2	PA3	B	K0
1	Vss (GND)	A	Vcc (+ 5 V)

Figura 3-5: Connettori di Applicazioni del KIM-1

22	Vss (GND)	Z	RAM/R,W
21	Vcc (+ 5)	Y	$\Phi 2$
20		X	PLL TEST
19		W	R/W
18		V	R/W
17	SST OUT	U	$\Phi 2$
16	K6	T	AB15
15	DB0	S	AB14
14	DB1	R	AB13
13	DB2	P	AB12
12	DB3	N	AB11
11	DB4	M	AB10
10	DB5	L	AB9
9	DB6	K	AB8
8	DB7	J	AB7
7	RST	H	AB6
6	NMI	F	AB5
5	RO	E	AB4
4	IRQ	D	AB3
3	$\Phi 1$	C	AB2
2	RDY	B	AB1
1	SYNC	A	AB0

Figura 3-6: Connettori di Espansione del KIM

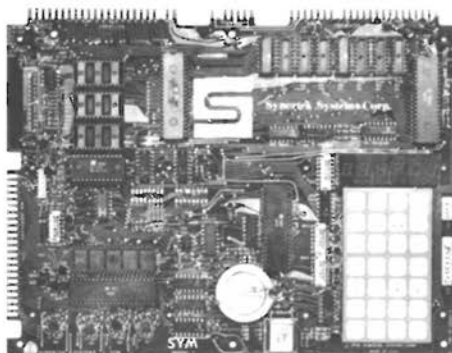


Figura 3-7: SYM

## IL SYM-1

La scheda SYM-1 fu introdotta dalla Synertek Systems come una versione espansa della scheda precedente. Una foto del SYM appare in Figura 3-7. In Figura 3-8, è riportata la sua organizzazione interna.

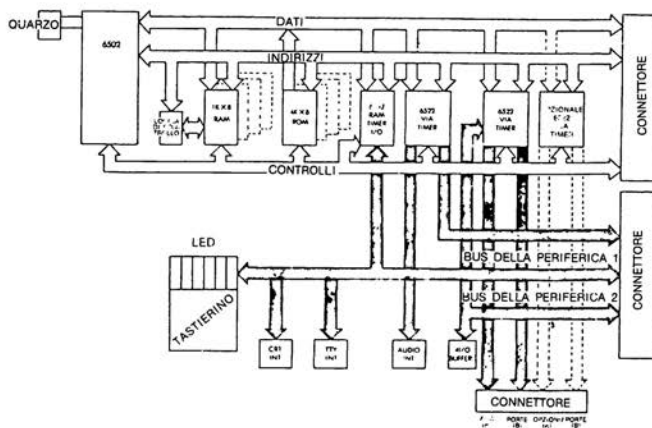


Figura 3-8: Organizzazione Interna del SYM

Le differenze essenziali dalla scheda precedente sono:

- È munito di una memoria ROM separata di 4 K per 8. Le maggiori dimensioni della ROM permettono l'impiego di un monitor residente più complesso.
- Comprende chip ingresso-uscita più complessi e ne ha tre invece di due, offre quindi più porte I/O e maggiori possibilità. A causa della porta in più possiede anche un connettore d'applicazione in più rispetto alla scheda precedente.
- Sono disponibili inoltre altre possibilità ingresso-uscita quali ad esempio quattro buffer ingresso/uscita e parte di un'interfaccia CRT.

Esistono altre differenze eterogenee fra questi sistemi ma non sono importanti ai fini di questo libro.

La mappa di memoria di sistema è mostrata in Figura 3-9; una più dettagliata mappa di memoria RAM, è riportata in Figura 3-10. I particolari dei tre connettori sono mostrati rispettivamente in Figura 3-11, 3-12 e 3-13.

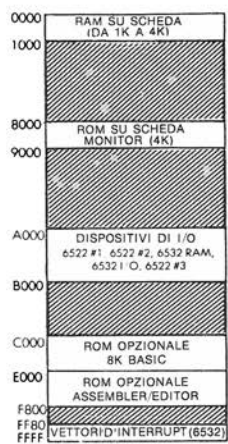


Figura 3-9: Mappa di Memoria del Sistema

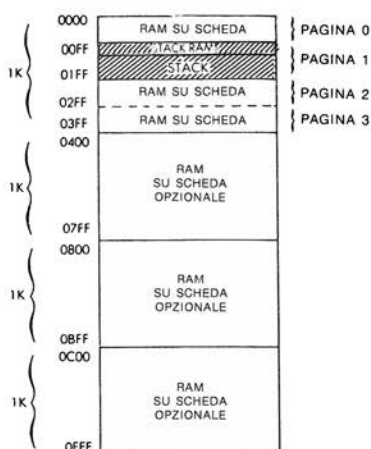


Figura 3-10: Mappa della Memoria RAM

1	SYNC	A	AB0
2	RDY	B	AB1
3	$\Phi 1$	C	AB2
4	$\overline{IRQ}$	D	AB3
5	RO	E	AB4
6	NMI	F	AB5
7	RES	H	AB6
8	DB7	J	AB7
9	DB6	K	AB8
10	DB5	L	AB9
11	DB4	M	AB10
12	DB3	N	AB11
13	DB2	P	AB12
14	DB1	R	AB13
15	DB0	S	AB14
16	$\overline{I8}$	T	AB15
17	DBOUT (1)	U	$\Phi 2$
18	POR	V	R/W
19	Non impiegato	W	R/W
20	Non impiegato	X	AUD TEST
21	+5V	Y	$\Phi 2$
22	GND	Z	RAM - R/W

Figura 3-11: Connettore di Espansione (E)

1	GND	A	+5V
2	APA3	B	$\overline{00}$
3	APA2	C	$\overline{04}$
4	APA1	D	$\overline{08}$
5	APA4	E	$\overline{0C}$
6	APA5	F	$\overline{10}$
7	APA6	H	$\overline{14}$
8	APA7	J	$\overline{1C}$
9	APB0	K	$\overline{18}$
10	APB1	L	Audio In
11	APB2	M	Audio Out (LO)
12	APB3	N	RCN-1 (1)
13	APB4	P	Audio Out (HI)
14	APA0	R	TTY KB RTN (+)
15	APB7	S	TTY PTR (+)
16	APB5	T	TTY KB RTN (-)
17	KB RIGA 0	U	TTY PTR (-)
18	KB COL F	V	KB RIGA 3
19	KB COL B	W	KB COL G
10	KB COL E	X	KB RIGA 2
21	KB COL A	Y	KB COL C
22	KB COL D	Z	KB RIGA 1

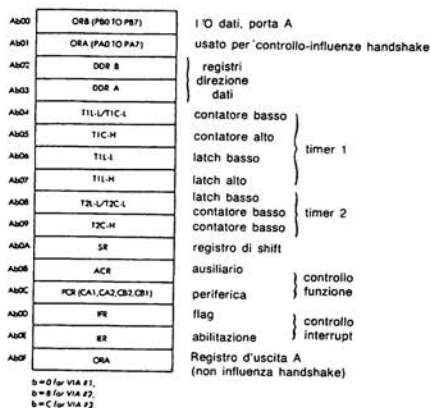
(1): Jumper Option

Figura 3-12: Connettore di Applicazione (A)

1	GND	A	+5V
2	-V <sub>N</sub>	B	+V <sub>p</sub>
3	2 PA 1	C	2 PA 2
4	2 CA 2	D	2 PA 0
5	2 CB 2	E	2 CA 1
6	2 PB 7	F	2 CB 2
7	2 PB 5	H	2 PB 6
8	2 PB 3	J	2 PB 4
9	2 PB 1	K	2 PB 2
10	2 PA 7	L	2 PB 0
11	2 PA 5	M	2 PA 6
12	2 PA 3	N	2 PA 4
13	RES	P	3 CA 1
14	3 CB 1	R	SCOPE
15	3 PB 2	S	3 PB 3
16	3 PB 0	T	3 PB 1
17	3 PA 6	U	3 PA 7
18	3 PA 3	V	3 PA 0
19	3 PA 4	W	3 PA 1
20	3 PA 5	X	3 PA 2
21	3 PB 5 (B)	Y	3 PB 4 (B)
22	3 PB 7 (B)	Z	3 PB 6 (B)

(B): Buffered

Figura 3-13: Connettore di Applicazione Ausiliario (AA)



3-14: Mappa di Memoria del 6522

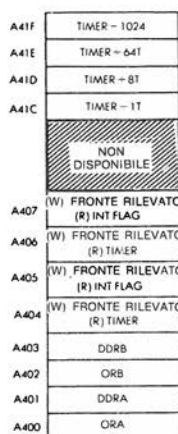


Figura 3-15: Mappa di Memoria del 6532

In Figura 3-14, è riportata la mappa di memoria del 6522, mentre in Figura 3-15 è mostrata la mappa di memoria del 6532.

Dal momento che alcuni dettagli di implementazione saranno usati (o sui quali si lavorerà) in alcuni dei programmi applicativi, vengono di seguito presentati due importanti particolari. La Figura 3-16 mostra le quattro uscite con buffer disponibili da PB4 a PB7 del 6522 #3. La Figura 3-17 mostra il collegamento ai LED ed alla tastiera.

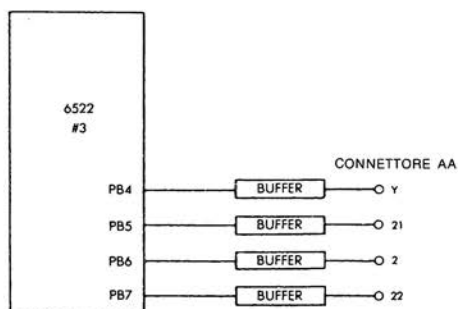


Figura 3-16: Le Quattro Uscite con Buffer



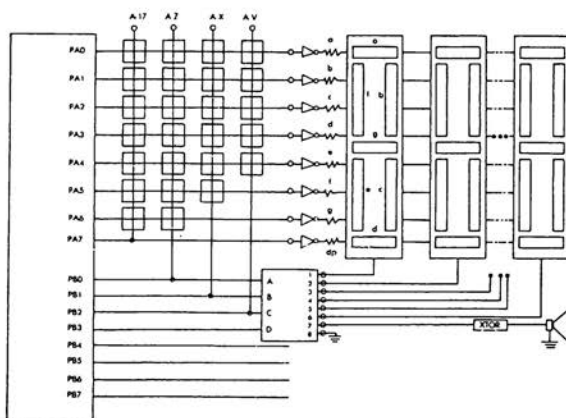


Figura 3-17: Collegamento della Tastiera e dei LED



Figura 3-18: AIM 65 è una Scheda con Mini-Stampante e Tastiera Completa

## L'AIM 65

L'AIM 65 è mostrato in Figura 3-18. Questa unità, sviluppata dalla Rockwell International, consiste di: due schede. Una di esse è il microcomputer, munito di una stampante a matrice di punti a 20 colonne ed un display alfanumerico a 20 caratteri. La seconda scheda è una tastiera completamente ASCII, la quale è connessa direttamente all'altra. La stampante opera a 120 linee al minuto, e impiega una matrice di punti cinque per sette per stampare l'insieme completo dei 64 caratteri ASCII (solamente nel caso peggiore). Nella sua versione minima l'AIM 65 è costituito da un monitor comprendente (8 K) 1 K di RAM, due 6522, un 6532, più le solite interfacce (telescrivente, due interfacce per cassette audio e naturalmente l'interfaccia per la tastiera). Diversi altri chip aggiuntivi possono essere posti sulla scheda. Inoltre il connettore d'applicazioni utente è identico a quello descritto per le precedenti schede. Un utente che sviluppa applicazioni per questa scheda specifica, dovrà modificare solamente i programmi qui presentati per adattare l'assegnazione di memoria del PIO dell'AIM 65.

## ALTRE SCHEDE

Altre schede vengono costruite da altri produttori quali ad esempio Ohio Scientific.

Ad ogni modo tutti i sistemi 6502 si adattano alla nostra descrizione di "sistema standard". Dal momento che essi usano gli stessi chip di I/O (e quasi tutti questi chip offrono grandi vantaggi), non ci sarà praticamente bisogno di fare modifiche ai programmi presentati in questo libro, fatta eccezione per l'indirizzamento del PIO e la possibile non disponibilità di particolari linee di I/O.

I connettori dei SYM A e E sono equivalenti a quelli dei KIM e dell'AIM. La scheda verticale sulla sinistra dell'alimentatore della Figura 3-19 sottoriportata, è una scheda di espansione di memoria di 16 K collegata attraverso il connettore E. In primo piano si vedono due prototipi collegati attraverso il connettore A: una tastiera esadecimale e una microstampante. Essi sono descritti nel capitolo 6.

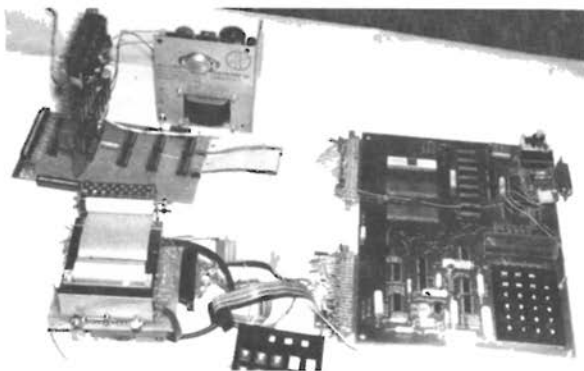


Figura 3-19: Compatibilità dei Connettori KIM/SYM/AIM

## CAPITOLO 4

# TECNICHE DI BASE

### INTRODUZIONE

In questo capitolo collegheremo un 6502 a dispositivi d'ingresso-uscita di base. Eseguiamo collegamenti con semplici dispositivi di uscita, quali diodi emettitori di luce (LED), relè ed altoparlanti. In ingresso collegheremo un set di Interruttori. Useremo queste risorse per iniziare lo sviluppo di semplici programmi applicativi, quali un generatore Morse, un orologio, un semplice programma per home control e anche una tastiera telefonica automatica. Presenteremo poi applicazioni dirette di queste tecniche: una sirena, un contatore d'impulso, un programma musicale, un gioco matematico. Nel capitolo successivo poi, svilupperemo programmi più complessi usando questi dispositivi d'ingresso-uscita di base e alcuni più complessi.

Sono necessari pochi componenti per realizzare in pratica le applicazioni di questo capitolo. Un'illustrazione del sistema è riportata in Figura 4-0. Tutti i componenti possono essere acquistati a basso costo in ogni magazzino di elettronica. Il lettore è vivamente incoraggiato ad acquistare questi pochi componenti elettronici e ad installarli come indicato in questo capitolo, con lo scopo di applicare effettivamente i programmi che saranno descritti. Naturalmente ciò richiederà l'accesso a sistemi basati sul 6502.

Con lo scopo di presentare programmi reali, viene usata una configurazione hardware sul SYM nella prima parte, e quella del KIM per la seconda. Ad ogni modo tutti questi programmi funzioneranno, con modifiche minime, su ogni altro sistema 6502 (vedi Capitolo 2).

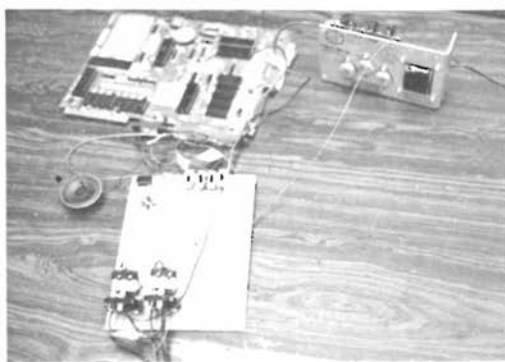
I programmi sviluppati in questo capitolo sono semplici, ma suppongono una conoscenza di base delle istruzioni del 6502, come quella fornita dal libro di questa serie, "Programmazione del 6502".

La seguente è la lista dei componenti richiesti per i programmi applicativi di questo capitolo:

scheda perforata	(1)
interruttori	(4)
LED driver	(1)
LED	(1 o più)
relè a 12 V	(3)
altoparlante	(1) (preferibilmente ad alta impedenza)
resistenza variabile	(1)
resistenze	
spina maschio per 120 V AC	(1)
spine femmine per 120 V AC	(2)

I collegamenti hardware dei diversi componenti sulla scheda saranno descritti per ogni applicazione.

Non è indispensabile eseguire l'assemblaggio di un'applicazione per comprendere questo capitolo. Comunque in questo e nel successivo capitolo saranno suggeriti molti esercizi. Sebbene essi possano essere sviluppati sulla carta, una migliore esperienza la si acquisisce trami-



*Figura 4-0:* Sistema Completo con Alimentatore, Microcomputer, Registratore a Nastro e Scheda Applicativa

te sperimentazione reale. Il lettore è quindi nuovamente incoraggiato, prima o dopo la lettura di questo libro, ad iniziare programmazione su hardware reale. Lo scopo di questo capitolo è di insegnare le tecniche base di interfacciamento hardware e software, che sono richieste per collegare ogni sistema 6502 "standard" a semplici dispositivi esterni. Alla fine di questo capitolo, saprete come sfruttare le principali risorse dei chip d'ingresso-uscita e come scrivere programmi che riconoscono e controllano dispositivi d'ingresso-uscita. Nel successivo capitolo ci baseremo su questa esperienza e svilupperemo applicazioni industriali ed home più complesse.

## PARTE 1: LE TECNICHE

### RELÈ

Per controllare un circuito esterno ad alta tensione e corrente, si impiega un relè: il circuito di controllo è *isolato* da quello esterno tramite il relè. Un relè richiede corrente continua. La corrente percorre la bobina producendo un campo magnetico. Tale campo provoca la chiusura di un contatto mobile. *Il circuito esterno può essere a corrente alternata (CA) o continua (CC).* Per ottenere il controllo di dispositivi esterni che impiegano una tensione e corrente piuttosto elevate, quali le applicazioni, useremo relè.

Nella scheda SYM ci sono speciali provvedimenti per altre correnti e tensioni. Sulla piastra sono disponibili quattro porte di uscita con buffer. Esse sono collegate rispettivamente ai bit 4, 5, 6 e 7 del registro B di Ingresso-uscita del PIO (6522-U29) (vedi Figura 4-1). Useremo quindi direttamente queste uscite speciali che possono controllare relè. Su ogni altra piastra che ha solamente uscite del PIO (come KIM), deve essere usato un transistor o un buffer. In Figura 4-2 è indicato l'impiego di un Invertitore Sestuplo per il controllo di tre relè esterni da due linee di uscita di un 6530.

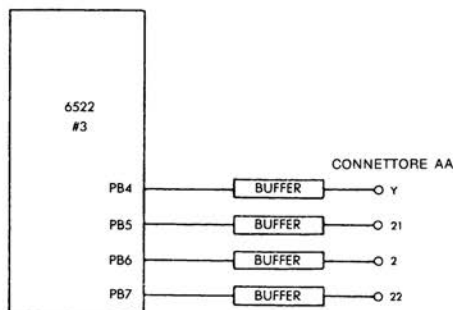


Figura 4-1: Buffer di I/O

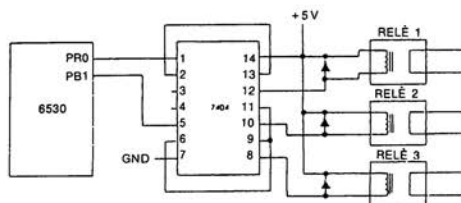


Figura 4-2: Interfaccia Relè del 6530

## L'Interfaccia Hardware

Lo schema per il collegamento di un singolo relè è indicato in Figura 4-3. Questo può essere, per esempio, un relè a 12 volt con una bobina da 50 o 500 ohm. Il contatto può essere SPST (singolo polo, singolo braccio = un contatto) o SPDT (singolo polo, doppio braccio = due contatti) a 10-15 ampere. La corrente sui contatti del relè deve essere sufficiente ad alimentare il dispositivo esterno ad esso collegato. La maggior parte delle applicazioni home non assorbono più di 10-15 ampere, cosicché le specifiche suddette saranno sufficienti in tali casi.

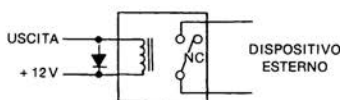


Figura 4-3: Collegamento di Un Solo Relè

Si noti che in parallelo alla bobina è collegato un diodo di clipping. Questa è una precauzione importante da adottare su ogni relè, per evitare danni al buffer del PIO o all'amplificatore. Quando il relè viene spento si fa un picco inverso di tensione. Ogni diodo che resiste a tale tensione va bene allo scopo. Per esempio, un IN914 sarà adatto ai nostri scopi.

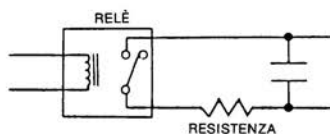


Figura 4-4: Protezioni Verso il Dispositivo

Dal relè verso il dispositivo bisogna prendere due precauzioni: in parallelo all'uscita può essere posto un condensatore per assorbire le scariche dovute alla chiusura dei contatti (questo assicura una vita più lunga ai contatti del relè; può essere messa anche una resistenza in serie se si può avere un assorbimento di corrente significativo). (vedi Figura 4-4).

Un relè a doppio-polo può essere collegato esattamente allo stesso modo e lo schema di collegamento è indicato in Figura 4-5. Un tale relè è in grado di servire contemporaneamente due circuiti separati ed indipendenti.

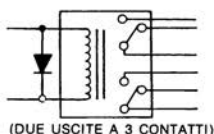


Figura 4-5: Collegamento di Un Relè a Doppia Via

Consideriamo ora un'applicazione pratica. Collegheremo due relè, rispettivamente R1 e R2, ai bit 6 e 7 della porta B del PIO del SYM. Questi relè saranno usati per controllare dispositivi CA. Nel caso più semplice, supporremo che questi dispositivi CA siano due lampade indipendenti. Questo ci permetterà di controllare facilmente il programma, verificando semplicemente se le lampade sono accese o spente. Naturalmente invece di una lampada, il dispositivo può essere di qualsiasi altro tipo che non sovraccarichi il relè. In Figura 4-6 appare lo schema dei collegamenti.

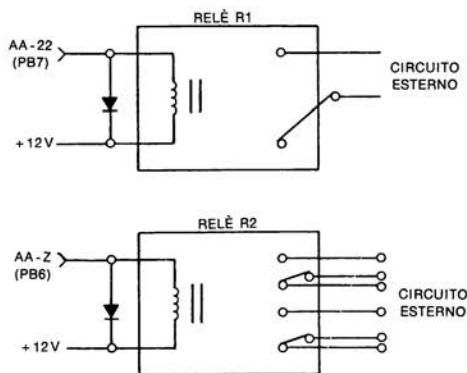


Figura 4-6: Collegamento di Due Relè al PIO

Esaminiamo le Figure 3-11, 3-12, 3-13 che riportano i punti di collegamento per i tre connettori SYM: si vede che le quattro uscite munite di buffer, indicate con PB4, PB5, PB6 e PB7, sono disponibili rispettivamente sui pin Y, 21, Z e 22. I punti di connessione segnati con PB5 fino a PB7 sulla nostra illustrazione, devono quindi essere semplicemente collegati con un filo agli appositi pin del «connettore ausiliario di applicazione».

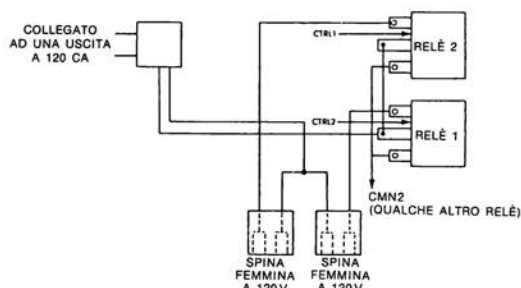


Figura 4-7: Circuiti Esterni per Relè

Dalla parte del relè verso il circuito esterno, è impiegata una spina CA la quale sarà collegata ad una presa della parete e fornirà potenza alle due uscite che saranno controllate dal microcomputer. Queste due uscite sono collegate ai relè come indicato in Figura 4-7. Esse sono alimentate in parallelo dalla spina CA. In ogni caso ognuna delle due prese d'uscita può essere alimentata indipendentemente sotto il controllo del microcomputer. Implementiamo ora il software di controllo di questi relè.

AC00	IOR-B
AC05	TIC-H
AC06	TIL-L
AC07	TIL-H
AC0B	ACR
AC0F	

Figura 4-8: Mappa di Memoria per 6522#3 (Terzo 6522 del SYM)



## L'Interfaccia Software

Ognuno dei due circuiti collegati ai relé R1 e R2, sarà alimentato quando viene attuato il relé corrispondente. Il relé verrà chiuso ponendo il corrispondente bit di controllo ad "1". Esaminando la Figura 4-8, si può vedere che la porta B per il 6522#3 è posta all'indirizzo di Memoria AC00. Il contenuto della suddetta locazione di memoria è mostrato in Figura 4-9. Ora chiudiamo ed apriamo i relé.

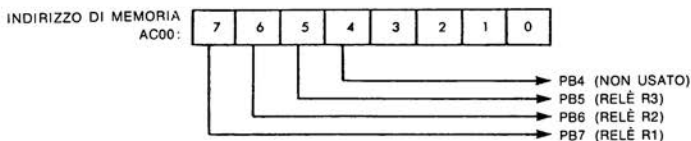


Figura 4-9: Porta B del 6522#3

Innanzitutto dobbiamo imporre alla porta B la configurazione d'uscita. Per semplicità imporranno ai bit da 0 a 7 di essere uscite, anche se qui useremo solo i bit 5, 6 e 7. Tale convenzione può essere cambiata in una convenzione diversa. Si ricorderà dal Capitolo 2, che per specificare la direzione in cui sono usate le linee di Ingresso-uscita, bisogna caricare con "0" od "1" il corrispondente bit del Registro Direzione Dati. Un uno nel Registro di Direzione Dati indicherà un'uscita. Uno zero indicherà un ingresso. Caricando tutti uno nel Registro di Direzione Dati si è sicuri che tutti i bit saranno usati come uscite.

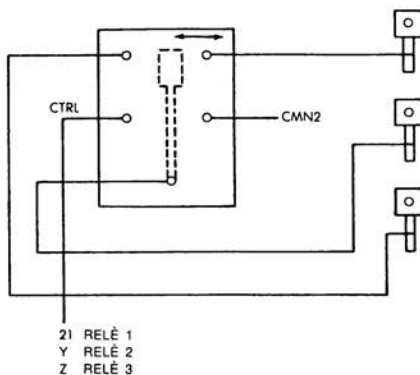


Figura 4-10: Particolare del Collegamento Relé sulla Piastra dell'Applicazione

Si noti che, durante la programmazione, un buon modo di procedere è di fare sempre cose il più possibile semplici e correnti. Dal momento che qui supporremo (come inizio) non esserci altri dispositivi collegati alle altre linee della porta B, non nuoce configurare tutte le linee come ingressi o come uscite.

La configurazione di tutti i bit programmati come uscite è ottenuta con le seguenti due istruzioni:

LDA	#\$FF	CARICA IN MODO IMMEDIATO A CON 11111111
STA	\$AC02	MEMORIZZA A NELL'INDIRIZZO AC02 ESADECIMALE

Sulla Figura 4-8 è possibile verificare che AC02 è l'indirizzo del Registro di Direzione Dati per la porta B del 6522#3.

L'esadecimale "FF" è equivalente al binario "11111111". Torniamo ora al relé collegato a PB6.

LDA	\$AC00	LEGGI IL VALORE CORRENTE DI PB
ORA	#\$40	METTI PB6 AD 1
STA	\$AC00	USCITA

La prima istruzione è usata per leggere il valore corrente della porta B. Poiché parecchi dispositivi o relé possono essere collegati alla porta B, non vogliamo scrivere semplicemente una maschera del tipo "01000000" nella porta B; ciò chiuderà infatti il relé collegato a PB6, ma aprirà anche tutti gli altri! Quindi leggeremo lo stato attuale di PB6 e cambieremo solo un bit, PB6. Tale cambiamento è eseguito tramite l'istruzione di OR logico, la seconda nel nostro programma (ORA). L'OR logico rispetta il valore degli altri bit e forza al valore "1" il bit specifico che interessa. Se si vuole porre ad "1" PB7 invece di PB6, sarà usata la maschera "80" (esadecimale), invece di "40". Infine, la maschera di bit risultante è memorizzata all'indirizzo AC00 che corrisponde a PB; il relé collegato a PB6 viene poi chiuso.

**ESERCIZIO 4-1:** Scrivere il programma di tre istruzioni, che chiuderà contemporaneamente i relé collegati a PB6 e PB7.

Apriamo ora i relé collegati a PB6:

LDA	\$AC00	LEGGI LO STATO CORRENTE DI PB
AND	#\$BF	METTI IL BIT 6 A 0
STA	\$AC00	MEMORIZZA IL VALORE RISULTANTE IN PB

L'istruzione di AND logico è usata per mettere a "0" un particolare bit. Tutti gli altri bit non sono influenzati. ("BF" esadecimale vale "10111111" in binario).

**NOTA:** L'istruzione di AND è tradizionalmente usata per mettere a zero uno specifico bit. Ad ogni modo lo stesso risultato può essere ottenuto usando l'istruzione EOR. Il programma resta lo stesso ad eccezione dell'istruzione AND che diventa:

EOR	#\$40	
-----	-------	--

Il vantaggio è che la maschera usata per l'accensione è uguale a quella per lo spegnimento. Questo elimina possibili errori. Il lettore potrà verificare che questo è un modo legittimo per forzare uno zero. Quanto sopra avviene poiché l'OR esclusivo di "1" e "1" è "0". Se il bit 6 era 1, la maschera "40" lo forzerà quindi a zero. Tutti gli altri bit saranno inalterati.

## Verifica

Verifichiamo ora, che queste semplici Istruzioni siano necessarie e sufficienti a chiudere ed aprire i nostri relé. Collegheremo due lampade, o due dispositivi, ai due relé e scriveremo queste istruzioni sulla tastiera, poi verificheremo che le lampade vengano accese e spente. Dal momento che la tastiera impone che i dati in ingresso siano in forma esadecimale, di seguito è riportato l'equivalente in esadecimale dei due programmi sopra:

Chiusura del relé	
AD	00 AC
09	MASCHERA (MASCHERA sta per un valore a 8bit)
8D	00 AC

Il programma per aprire il relé è:

AD	00 AC
49	MASCHERA
8D	00 AC

Se si possiede una scheda microcomputer, ora si potranno inserire questi due programmi e controllare il loro corretto funzionamento.

## INTERRUTTORI

Possono essere collegati due principali tipi di interruttori: un pulsante (interruttore SPST) o un interruttore a due posizioni (SPDT). Il collegamento di un SPST è illustrato in Figura 4-11. Nel collegamento indicato l'interruttore è nello stato logico "1" quando il contatto è aperto e nello stato "0" quando è chiuso. Se si desidera l'opposto basta semplicemente invertire le polarità sui contatti dell'interruttore.

In Figura 4-12 è indicato il collegamento di un interruttore SPDT (interruttore a due posizioni). Il collegamento è diretto. Una delle posizioni del contatto è allo stato logico "0".

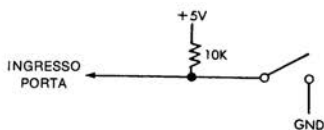


Figura 4-11: Collegamento di Un SPST

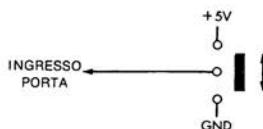


Figura 4-12: Collegamento di Un SPDT

## Collegamento di Quattro Interruttori

Useremo le linee 1, 2, 3 e 4 della porta B del 6522 come quattro ingressi impiegati per rilevare lo stato degli interruttori esterni. Il circuito in esame appare in Figura 4-13. Esaminiamo il programma.

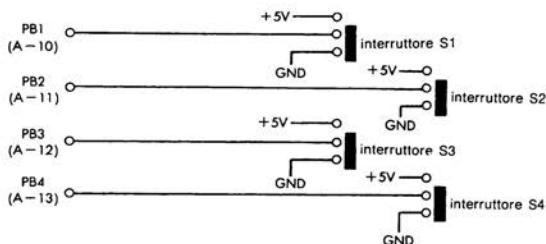


Figura 4-13: Collegamento di Quattro Interruttori SPDT alSYM

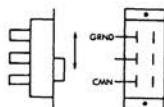


Figura 4-14: Un Interruttore SPDT

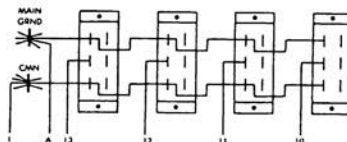


Figura 4-15: Particolare Collegamento per Quattro SPDT

## L'Interfaccia Software

Dobbiamo innanzi tutto dare la configurazione di linee di ingresso alle linee PB1, PB2, PB3 e PB4 della Porta B. Questo lo si esegue caricando la maschera opportuna all'indirizzo "A002", il registro di direzione dati della Porta B.

```
LDA    #$E0      PONI I BIT 01234 COME INGRESSI
STA    $A002
```

Per condizionare le linee 0, 1, 2, 3, 4 come ingressi e le linee 5, 6 e 7 come uscite si usa la

maschera "E0" (le linee di uscita possono essere collegate a relé esterni). "E0" in esadecimale vale "11100000" in binario. Ogni "0" indica un ingresso. Ogni "1" indica un'uscita. Può anche essere usato "E1".

Ora leggeremo il valore di un interruttore e salteremo a una locazione di memoria specifica determinata da questo valore.

LDA	#SWITCHPTR	"02" PER S1, "04" PER S2, "08"
		PER S3, "10" PER S4
BIT	\$A000	A000 È L'INDIRIZZO
BEQ	ANY ADDRESS	SALTA ALL'INDIRIZZO INDICATO SE L'INTER-
		RUTTORE ERA ZERO (OFF).

In alternativa, se si desidera saltare ad una specifica locazione di memoria se l'interruttore è "1" (ON), bisognerà sostituire l'istruzione BNE con la BEQ nell'ultima riga del programma.

### Prova del Programma sulla scheda

Il codice esadecimale per il programma di cui sopra è il seguente:

A9	SWITCHPTR
2C	00 A0
FO	ANYADDRESS O "DO" ANYADDRESS

## ALTOPARLANTE

Un altoparlante esterno può essere direttamente collegato a un pin di uno dei PIO. Spesso il pin 7 è il più potente ed è quindi impiegato allo scopo. Sul 6522 la polarità del segnale d'uscita di PB7, può essere controllata da uno dei timer d'intervallo interni. Il timer sarà usato per generare un tono di frequenza data. La posizione migliore per collegare l'altoparlante sarà quindi PB7. In Figura 4-16 è mostrato lo schema del collegamento.

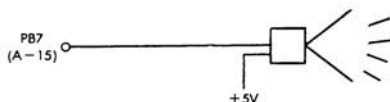


Figura 4-16: Collegamento dell'Altoparlante

Quando si usa l'uscita con buffer del SYM (6522#3), bisogna porre una resistenza in serie con l'altoparlante per limitare la corrente d'uscita. Invece di collegare l'altoparlante direttamente a un pin d'uscita del PIO, si può usare il circuito di Figura 4-17 per ottenere un suono più forte.

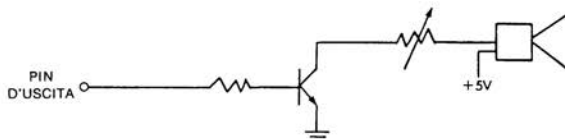


Figura 4-17: Realizzazione di un'Uscita più Alta

*Attenzione:* In Figura 4-17 è riportata una resistenza variabile per comodità. Ad ogni modo, se si pone a zero tale resistenza, probabilmente si brucerà e si distruggerà il corrispondente transistor d'uscita (cioè si usa anche sul SYM).

## L'Interfaccia Software

Un suono con l'altoparlante può essere generato semplicemente accendendolo e spegnendolo alla frequenza desiderata. Il suono non sarà "pulito" come quello di uno strumento musicale dal momento che è stato generato con un'onda quadra. In ogni caso sarà sufficiente per le nostre esigenze e può chiaramente essere identificato dalla sua frequenza. Costruiremo ora un'applicazione pratica.

## UN GENERATORE MORSE

Svilupperemo un programma in grado di generare un codice Morse corrispondente a ciascuna lettera dell'alfabeto. Il programma attiverà un altoparlante, cosicché si potrà verificare che si sta generando il codice Morse adeguato. Inoltre, esso sarà in grado di accendere o spegnere un dispositivo esterno cosicché il codice Morse potrà per esempio essere trasmesso su una rete di comunicazioni.

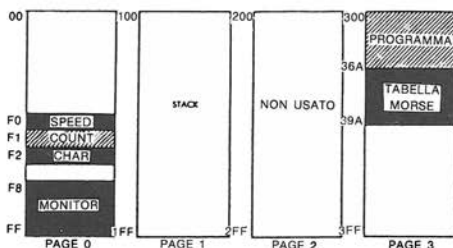


Figura 4-18: Assegnazione di Memoria per il Programma Morse

Le convenzioni usate in questo programma sono le seguenti: Il programma sarà memorizzato a Pagina 3 della RAM, cioè, inizierà alla locazione 300. Tale fatto è illustrato in Figura 4-18. Il programma contiene una tabella di equivalenza Morse che servirà per generare la maschera di bit giusta per ogni dato carattere ASCII. In seguito si mostrerà come viene generata questa tabella. Si fa l'ipotesi che il primo carattere da convertire in Morse sia contenuto nell'accumulatore alla partenza del programma.

La velocità di trasmissione sarà inoltre regolabile, tramite la variabile SPEED, memorizzata in Pagina 0 alla locazione di memoria F0 (vedi Figura 4-18). Ogni unità di tempo (quale, ad esempio, la durata di un tempo in codice Morse) è espressa internamente in millisecondi. Ponendo a 100 il valore della variabile "SPEED", si otterrà una durata di 1/10 di secondo.

Prima dell'inizio del programma, si suppone che CHAR e SPEED abbiano un contenuto valido e che l'accumulatore contenga il primo carattere che deve essere trasmesso. Una subroutine esterna potrà chiamare questa subroutine ripetitivamente, al fine di trasmettere una stringa di caratteri. È compito di questa subroutine depositare un carattere nell'accumulatore ogni volta che essa chiama il trasmettitore Morse.

Esaminiamo ora l'algoritmo impiegato per trasmettere il codice Morse.

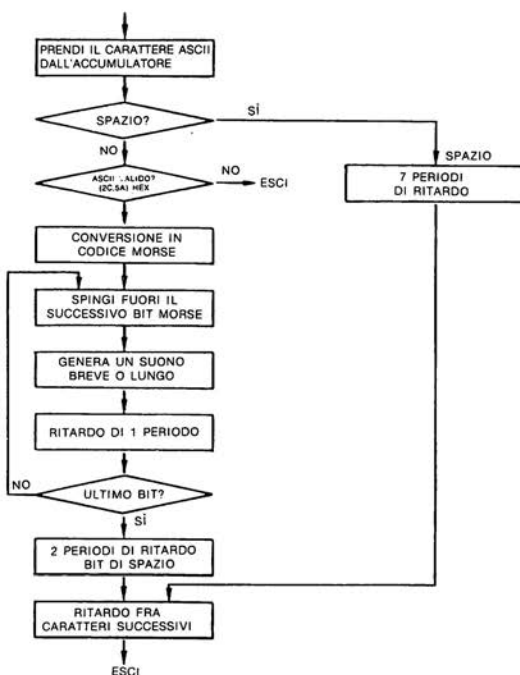


Figura 4-19: Diagramma di Flusso della Trasmissione Morse

Questo algoritmo è illustrato nel diagramma di flusso (flow-chart) di Figura 4-19. Il programma inizialmente controlla se c'è un carattere di spazio. Se lo trova, non genera alcun segnale per sette periodi, più il ritardo fra i caratteri successivi.

Si controlla poi che il carattere ASCII contenuto nell'accumulatore abbia un codice esadecimale valido. I codici consentiti devono essere compresi tra "2C" e "5A" in esadecimale (assumendo un codice ASCII a 7-bit). Altrimenti si ha come esito un errore. Dopo aver convalidato il codice del carattere, bisogna convertire il codice ASCII nel suo equivalente Morse. Tale tecnica sarà spiegata più avanti.

La codifica binaria del codice Morse consisterà in un bit di "START" (un "1"), seguito da uno "0" per un ".", e un "1" per una "-". Tutti i bit non utilizzati all'interno della parola di 8 bit saranno posti a "0". Questa conversione sarà eseguita dal programma con una tabella di look-up (consultazione) descritta più avanti. Supponiamo ora che si sia ottenuta la conversione in binario del codice Morse. Bisogna generare la sequenza di toni. Il contenuto dell'accumulatore sarà sottoposto a uno shift fino a che non si trova START. Successivamente al riconoscimento del bit di START, ogni "0" sarà interpretato come ".", e ogni "1" sarà interpretato come "-" fino all'ottavo bit. Per ogni "0" si genererà un tono breve. Per ogni "1" si genererà un tono lungo.

Anche la generazione del tono sarà descritta in dettaglio più avanti.

Dopo la generazione di un tono corrispondente ad un bit, viene inserito un periodo di attesa e il bit successivo del codice Morse è controllato fino a che è stato trovato l'ultimo (ottavo). Dopo la trasmissione della sequenza di toni per un carattere Morse, vengono generati due periodi di ritardo. Questi corrispondono ai bit di "spazio" che sono normalmente inseriti alla fine della trasmissione di ogni carattere. Si genera poi un periodo di ritardo per separare i caratteri successivi.

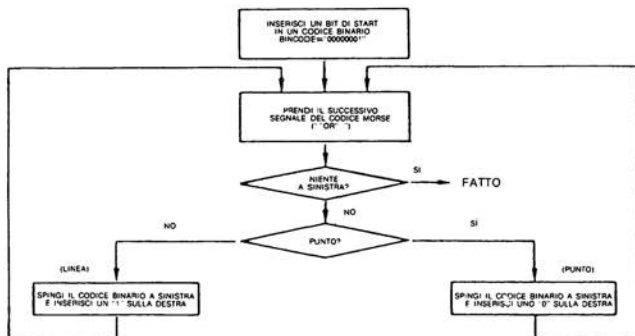


Figura 4-20: Conversione da Morse a Binario

La sequenza è illustrata in modo chiaro nel diagramma di flusso di Figura 4-20 e può essere verificato dal lettore. Esaminiamo ora in dettaglio i problemi specifici che non abbiamo ancora risolto.

## Conversione da ASCII a Morse Binario

Vogliamo ora stabilire una tabella di corrispondenza fra i caratteri ASCII e la rappresentazione in binario del loro codice Morse. Spieghiamo ciò con un esempio.

Il carattere "B" ha il codice Morse "...-..".

Ogni "-" sarà codificata con un "1" e ogni "." con uno "0". Il binario equivalente di "...-.." è quindi "1000".

Inoltre, per convenzione, aggiungeremo un bit di START (un "1") sulla sinistra del codice che abbiamo appena generato. Il codice risultante a questo punto è: "11000". Infine ogni codice Morse in binario sarà contenuto in una parola a 8 bit. I rimanenti bit alla sinistra del bit di START saranno ora posti a zero. Il nostro codice finale a 8 bit è quindi: "00011000". In esadecimale il suo valore è "18".

La rappresentazione decimale della codifica binaria Morse di B è: "18".

La seguente tabella mostra come esempio gli esadecimali equivalenti di A, B, C, D. Una tabella completa di equivalenza per tutti i caratteri consentiti dal codice Morse, è riportata in Figura 4-22. L'algoritmo corrispondente alla tecnica appena descritta, è illustrato dal diagramma di flusso di Figura 4-23.

Abbiamo ora stabilito una tabella di equivalenza per tutti i caratteri ASCII. Tale tabella sarà chiamata "Tabella Morse" e sarà memorizzata alla fine del programma (vedi Figura 4-18).



ando richiederemo il codice Morse equivalente di uno specifico carattere, accederemo alla tabella e vi troveremo il codice binario. Ciò sarà descritto più avanti nel capitolo 4.2.2. In questo capitolo, invece, ci occuperemo di come implementare il programma in questione.

Lettera	ASCII	Morse	Binario	Esadecimale
A	41	.-	00000101	05
B	42	-...-	00011000	18
C	43	-.-.	00011010	1A
D	44	...-	00001100	0C

Figura 4-21: Conversione da ASCII a Morse

Carattere	Codice Morse	ASCII	Tabella dei valori in Esadecimale
.	...	2C	73
-	---	2D	31
.	...	2E	55
.	...	2F	32
0	----	30	3F
1	..---	31	2F
2	...---	32	27
3	....--	33	23
4	.....	34	21
5	-----	35	2
6	-----	36	3
7	-----	37	38
8	-----	38	3C
9	-----	39	3E
:	Definibili dall'utilizzatore	3A	1
:	Definibili dall'utilizzatore	3B	1
<	Definibili dall'utilizzatore	3C	01
=	Definibili dall'utilizzatore	3D	1
>	Definibili dall'utilizzatore	3E	01
?	Definibili dall'utilizzatore	3F	4C
	Definibili dall'utilizzatore	40	01
A	.-	41	05
B	-...-	42	18
C	-.-.	43	1A
D	...-	44	0C
E	..	45	02
F	....	46	12
G	...-	47	0E
H	....	48	1C
I	..	49	04
J	.-.-.	4A	17
K	-. -	4B	0D
L	..-.	4C	14
M	---.	4D	07
N	---.	4E	06
O	---	4F	0F
P	..-..	50	16
Q	-.-.-	51	1D
R	..-.	52	0A
S	...-	53	08
T	..-	54	03
U	....	55	09
V	...-	56	11
W	..-.-	57	0B
X	-. -	58	19
Y	-. -	59	1B
Z	---.	5A	1C

Figura 4-22: Tabella delle Equivalenze Morse

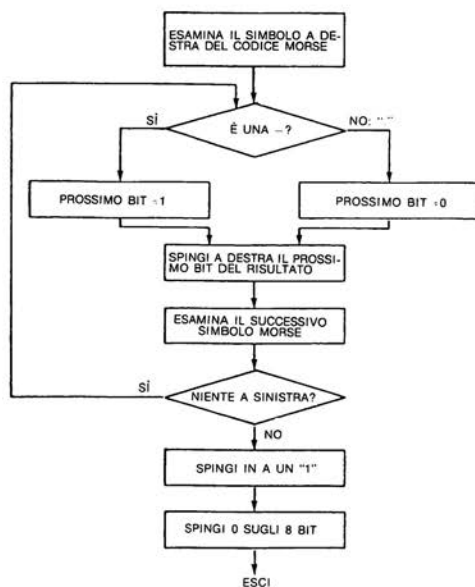


Figura 4-23: Diagramma di Flusso per la Generazione del Codice Morse in Esadecimale

## Generazione di un Tono con il Timer

Il prossimo problema è quello di generare un tono di frequenza e durata stabilita. In questo caso useremo un timer.

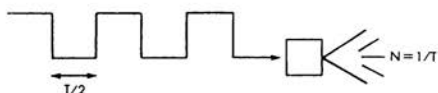


Figura 4-24: L'Onda Quadra Genera un Tono nell'Altoparlante

Il tono sarà generato inviando all'altoparlante un'onda quadra della frequenza richiesta. Ciò è illustrato in Figura 4-24. Il timer può essere usato per generare automaticamente questa forma d'onda. Per ottenere tale risultato posizioneremo il bit opportuno del registro di controllo

ACR (vedi Figura 4-25), poi controlleremo semplicemente la durata di questo tono o forma d'onda. Il diagramma di temporizzazione è riportato in Figura 4-26. Ø2 nella parte alta dell'illustrazione è la fase 2 del clock del sistema. Nella maggior parte dei sistemi 6502 standard il clock ha il periodo di un microsecondo. L'impulso generato con il timer viene fornito sul pin d'uscita PB7. Esso sarà almeno  $N + 1,5$  sottocicli, dove  $N$  è il valore inserito nel contatore. Questo perché il contatore del timer viene decrementato da  $N$  a 0 ed inverte la porta d'uscita con la successiva transizione alto-basso del clock. Tutto ciò è illustrato in Figura 4-26. Allo stesso istante viene anche generato un interrupt (IRQ), ma qui non sarà usato.



Figura 4-25: Registro Ausiliario del 6522

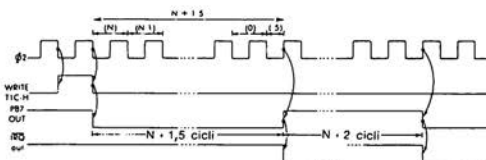


Figura 4-26: Diagramma di Temporizzazione per la Generazione del Tono

Per l'impiego del timer si deve, quindi, depositare un valore opportuno nel suo contatore. Non appena il contenuto del contatore viene inserito questo inizia la sua corsa. Dal momento che il contatore è un registro a 16 bit non può essere caricato dal microprocessore con un trasferimento di dato singolo. *Esso deve subire un latch*. Il timer è quindi munito di un latch interno a 16 bit chiamato T1L. La parte inferiore del latch è indicata con T1L-L, mentre la parte superiore è indicata con T1L-H. Il valore  $N$  sarà inserito in T1L-L e T1L-H. A questo punto il contenuto a 16 bit è stato specificato ma non avviene nulla. Per far partire il timer bisognerà dare uno speciale ordine che trasferirà il contenuto del latch nel contatore attuale. Tale ordine è "scrivi T1C-H" che appare nella quarta riga di Figura 4-27:

LDA	#VALUE LO	
STA	\$A006	CARICA LA PARTE BASSA DEL LATCH
LDA	#VALUE HI	
STA	\$A007	CARICA LA PARTE ALTA DEL LATCH
STA	\$A005	PONI LATCH=START

Figura 4-27: Programma per l'Impiego del Timer 1



Figura 4-28: Generazione di un Tono di Durata Stabilito con il Timer 1

La sequenza di eventi per l'impiego del timer è stata ora chiarita. Essa è descritta dal diagramma di flusso di Figura 4-28. Innanzi tutto metteremo l'apposito bit del registro di controllo ACR al valore richiesto. Il timer funziona nel modo "free-running" generando un'onda quadra su PB7. Ciò lo si ottiene ponendo i bit 6 e 7 di ACR a "0" e "1" (vedi Figura 4-29 e 4-30). Fatto ciò bisogna memorizzare nel latch l'apposito valore N. Poi tale valore sarà trasferito nel contatore per farlo partire. Questo sarà il momento in cui si inizia a generare il tono. Ogni volta che il contatore arriverà a zero, ricaricherà automaticamente il valore memorizzato nel suo registro di latch. Il timer genererà quindi da quel momento un'onda quadra con un semiperiodo pari a circa  $N+2$  (questo è approssimato a causa del fatto che la parte bassa dell'impulso, ha una durata pari a  $N+1,5$ , mentre la parte superiore ha una durata di  $N+2$ ).

ACR 7 ABILITA- ZIONE USCITA	ACR 6 ABILITA- ZIONE FREE RUN	MODO
0	0 (ONE-SHOT)	Genera INT di time out quando è caricato T1 PB7 disabilitato.
0	1 (FREE RUN)	Genera INT continui PB7 disabilitato.
1	0 (ONE-SHOT)	Genera INT e impulso di uscita su PB7 ogni volta che è caricato T1. one-shot e larghezza d'impulso programmabile.
1	1 (FREE RUN)	Genera INT continui e onda quadra in uscita su PB7.

Figura 4-29: Selezione dei Modi del Timer con l'ACR del 6522

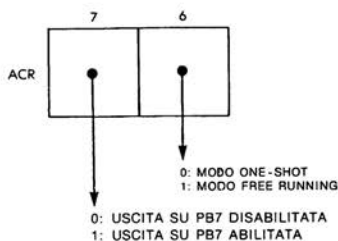


Figura 4-30: Bit 6 e 7 del ACR

LINE	*LOC	CODE	LINE
0002	0000		.QUESTA È UNA SUBROUTINE CHE ACCETTA I CARATT ASCII
0003	0000		.DA 2CH A 5AH (PIÙ 20H PER SPAZIO) E SUONA
0004	0000		.IL COD MORSE EQUIV SU UN ALTOPARLANTE COLLEGATO A
0005	0000		.PB7. 6522-025. INOLTRE ESSO COMMUTA ON OFF PB0. 6522-
0006	0000		.025 E. CON UN DRIVER, QUESTO BIT PUÒ AZIONARE UN
0007	0000		.TRASMETTITORE. UN PROG PRINC. CHIAMERÀ QUESTA SUB
0008	0000		.CON IL CARATT. ASCII IN ACC.
0009	0000		.ESEMPL SI POSSONO AVERE
0010	0000		.ACCETTANDO INGRESSI DA UN TERMINALE PER AVERE IL CODICE MORSE
0011	0000		.CON QUESTO Progr. OPPURE GENERARE UN ORDINE CASUALE
0012	0000		.DI CARATT. PER UN CODICE
0013	0000		.IL FORMATO DEI CARATT. DEL COD. MORSE DELLA TABELLA
0014	0000		.E DA SINISTRA A DESTRA. IL PRIMO BIT
0015	0000		.UGUALE AD 1 È IL BIT DI START ED OGNI
0016	0000		.1 SEG. È UNA LINEA ED OGNI 0 UN PUNTO
0017	0000		SPEED - SF0
0018	0000		COUNT - SF1
0019	0000		CHAR - SF2
0020	0000		= \$300
0021	0300	C9 20	MORSE CMP # \$20 ;SE UNO SPAZIO, ESEGUI LA ROUTINE DI SPAZIO
0022	0302	F9 07	BEG SPACE
0023	0304	C8 2C	CMP # \$2C ;VEDI SE IL CODICE ASCII
0024	0306	90 4E	BCC EXIT ; È MINORE DI 2CH. E RITORNO SE COSÌ
0025	0308	C9 5B	CMP # \$5B ;VEDI SE IL CODICE ASCII È MAGGIORE
0026	F10A	B0 4A	BCC EXIT ; DI 5AH E RITORNO SE COSÌ
0027	030C	AA	TAX ;PONI IL CODICE NEL REGISTRO INDICE
0028	030D	BD 45 03	LDA TABLE - \$2C.X ;ACCETTA IL CARATTERE MORSE
0029	0310	A0 08	LDY # \$8 ;NUMERO DI BIT DA RUOTARE L'ACCUMULATORE
0030	0312	84 F1	STY COUNT
0031	0314	0A	STARTB ASL A
0032	0315	C6 F1	DEC COUNT
0033	0317	90 FB	BCC STARTB ;FA SCORRERE FINCHÉ NON TROVI IL BIT DI START
0034	0319	85 F2	STA CHAR
0035	031B	A5 F2	LDX CHAR
0036	031D	0A	ASL A ;ORA EMETTI IL CODICE MORSE (1 - LINEA, 0 - PUNTO)
0037	031E	85 F2	STA CHAR
0038	0320	A0 01	LDY # \$1 ;PUNTO - 1 PERIODO DI TEMPO. SALTA AL PUNTO
0039	0322	90 02	BCC SEND ;E CARRY È 0, PUNTO
0040	0324	A0 03	LDY # \$3 ;ALTRIMENTI LINEA (3 PERIODI DI TEMPO)
0041			.QUESTA PARTE INVIA UN'USCITA ALTA PER N DI PERIODI DI TEMPO (REG. Y1 ED
			; UNUSI ITA BASSA PER 1 PERIODO
			SEND LDA # \$C0
0043	0326	A9 C0	STA \$A00B ;PONI IL MODO DEL TIMER A FREE RUNNING PER
0044	0328	8D 0B A0	STA \$A00B ; QUESTO VALORE
0045	0329	A9 00	LDA # \$0
0046	032D	8D 06 A0	STA \$A006
0047	0330	A9 04	LDA # \$04 ;E QUESTO VALORE DETERMINA IL TONO
0048	0332	8D 07 A0	STA \$A007 ; DELL'USCITA (CIRCA 1000 MHz)
0049	0335	8D 05 A0	STA \$A005 ;QUESTO FA PARTIRE IL TONO
0050	0338	A9 01	LDA # \$1 ;PONI AD 1 IL BIT D'USCITA PB0
0051	033A	8D 00 A0	STA \$A000
0052	033D	20 57 03	JSR DELAY ;RITARDO PER UN PERIODO DI TEMPO
0053	0340	A9 00	LDA # \$0
0054	0342	8D 0B A0	STA \$A00B ;PONI A 0 TONF
0055	0345	8D 00 A0	STA \$A000 ;PONI A 0 IL BIT D'USCITA (PB0)

Figura 4-31: Il Programma Morse (Listing completo in Appendice C)

```

0056 0348 A0 01 LDY # $01
0057 034A 20 57 03 JSR DELAY ;RITARDO PER 1 PERIODO DI TEMPO (SPAZIO TRA ELEMENTI)
0058 034D C6 F1 DEC COUNT ;DECREMENTA CONTEGGIO - VEDI SE SONO RUOTATI 8 BIT
0059 034F D0 CA BNE NEXT ;SE NO ESEGUI UN ALTRO ELEMENTO
0060 0351 A0 02 FINISH LDY # $2 ;RITARDO PER 3 PERIODI TEMPO (2 QUI PIÙ LO SPAZIO
0061 0353 20 57 03 JSR DELAY ; ALLA FINE DELL'ULTIMO ELEMENTO)
0062 0356 60 EXIT RTS
0063 ; RITARDO PER (REG. Y) * SPEED * 0.005 SEC
0064 0357 98 DELAY TYA
0065 0358 0A ASL A
0066 0359 0A ASL A
0067 035A A8 TAY
0068 035B A5 F0 D3 LDA SPEED
0069 035D A2 FA D2 LDX # $FA
0070 035F CA D1 DEX
0071 0360 D0 F0 BNE D1
0072 0362 38 SEC
0073 0363 E9 01 SBC # $1
0074 0365 D0 F6 BNE D2 ;RITARDO PER 7 PERIODI DI TEMPO
0075 0367 88 DEY ; (SPAZIO TRA PAROLE)
0076 0368 D0 F1 BNE D3 ;RITORNO DAL PROGRAMMA MORSE
0077 036A 60 RTS
0077 036B A0 07 SPACE LDY # $7
0077 036D 20 57 03 JSR DELAY
0077 0370 60 RTS
0077 0371 73 TABLE ;BYTE $73, $31, $6A, $32, $3F, $2F
0077 0372 31
0078 0373 6A
0078 0374 32
0078 0375 3F
0078 0376 2F ;.BYTE $27, $23, $21, $20, $30, $38
0078 0377 27
0078 0378 23
0078 0379 21
0078 037A 20
0078 037B 30
0078 037C 38
0078 037D 3C ;.BYTE $3C, $3E, $01, $01, $01, $01
0078 037E 3E
0080 037F 01
0080 0380 01
0080 0381 01 ;.BYTE $01, $4C, $01, $05, $18, $1A
0080 0382 01
0080 0383 01
0080 0384 4C
0081 0385 01
0081 0386 05
0081 0387 18
0081 0388 1A
0081 0389 0C ;.BYTE $0C, $02, $12, $0E, $10, $04
0081 038A 02
0082 038B 12
0082 038C 0E
0082 038D 10
0082 038E 04
0082 038F 17 ;.BYTE $17, $0D, $14, $07, $06, $0F
0082 0390 0D
0083 0391 14
0083 0392 07
0083 0393 06
0083 0394 0F
0083 0395 16 ;.BYTE $16, $1D, $0A, $06, $03, $09
0083 0396 1D
0084 0397 0A
0084 0398 08
0084 0399 03
0084 039A 09
0084 039B 11 ;.BYTE $11, $0B, $19, $1B, $1C
0085 039C 0B
0085 039D 19
0085 039E 1B
0085 039F 1C

SYMBOL TABLE
SPEED 00F0 COUNT 00F1 CHAR 00F2
MORSE 0300 STARTR 0314 NEXT 031B
SEND 0326 FINISH 0351 EXIT 0356
DELAY 0357 D3 035B D2 035D
D1 035F SPACE 036B TABLE 0371

```

Figura 4-31: Il Programma Morse (Continua)

Il tono deve agire durante un periodo chiamato "DELAY". La durata di questo ritardo può essere ottenuta tramite tecniche hardware o software. Infine il tono deve essere spento quando si è raggiunto il ritardo richiesto. Ciò lo si esegue ponendo a "0" il bit 7 di ACR.

Il lettore può fare riferimento al flow-chart di Figura 4-28 per la comprensione della sequenza di azioni necessarie per l'uso del timer. L'implementazione attuale sarà presentata in seguito con il programma.

## Il Programma Morse

Seguiremo qui il flow-chart che è stato presentato in Figura 4-19 e svilupperemo il corrispondente programma. In tale programma saranno usate un certo numero di tecniche specifiche:

*Indirizzamento indicizzato* sarà usato per richiamare la codifica in binario del codice Morse di un dato carattere ASCII.

*Il timer hardware* sarà usato per generare un tono di frequenza fissata. Sarà implementato un ritardo software per regolare la durata del tono.

*Loop nested* (cicli annidati) saranno usati per implementare una moltiplicazione nel loop di ritardo.

Esaminiamo ora il programma. Esso presuppone che l'accumulatore sia stato caricato con il valore del carattere ASCII che deve essere trasmesso in codice Morse (vedere la mappa di memoria di Figura 4-18). Per avere flessibilità, la velocità di trasmissione è regolabile. Essa è espressa in unità di 1 millisecondo (0,001 secondi). La variabile "SPEED" nella locazione di memoria "00FA" deve essere decisa prima di entrare in questo programma. Per esempio, se "SPEED" è posta al valore 1000, la durata di un "." sarà  $1000 \times 0,001 = 1$  secondo. Il programma risiederà a Pagina 3, con inizio all'indirizzo "0300" esadecimale.

L'inizio del programma è:

```
SPEED = $00F0
COUNT = $00F1
CHAR = $00F2
* = $0300
```

Le prime quattro linee sono istruzioni assembler. Le prime tre istruzioni assegnano rispettivamente gli indirizzi di memoria 00F0, 00F1, 00F2 a SPEED, COUNT e CHAR. La quarta istruzione, indica essere 0300, esadecimale, il valore del pseudo contatore-indirizzo, cioè, indica che la prima istruzione eseguibile nel programma sarà posta all'indirizzo di memoria 0300.

Si deve innanzi tutto controllare che il carattere nell'accumulatore sia un codice consentito. Ciò è compiuto con le seguenti istruzioni:

MORSE	CMP	#\$20	È UN CODICE DI SPAZIO?
	BEQ	SPACE	
	CMP	#\$2C	ERRORE SE MINORE DI 2C
	BCC	EXIT	
	CMP	#\$5B	O MAGGIORE DI 5B
	BCS	EXIT	

Le prime due linee controllano se il carattere nell'accumulatore è un carattere di "spazio" (20 esadecimale). Se il controllo dà risultato affermativo, viene introdotto un ritardo di sette periodi seguito dal normale ritardo fra due caratteri.

Le successive quattro istruzioni verificano che il codice ASCII sia compreso fra "2C" e "5A". Questo è il campo dei caratteri ASCII validi per la trasmissione Morse. Se si trova un carattere non consentito, si è rilevato un errore, si ha un salto alla locazione "EXIT". Allo scopo di mantenere il programma semplice ed educativo, non viene fatta nessuna azione specifica, alla locazione EXIT, per segnalare l'errore. Il lettore è incoraggiato (per esercizio) ad aggiungere i-

struzioni specifiche alla locazione EXIT, le quali segnalino il carattere sbagliato trovato nell'accumulatore. In questo programma il carattere sbagliato non viene semplicemente trasmesso.

Non appena si trova, in accumulatore, un carattere ASCII consentito, esso deve essere convertito in codice binario, il quale sarà usato per generare una sequenza di suoni. Il codice Morse binario corrispondente a ogni possibile carattere ASCII, è memorizzato alla fine del programma, dalla locazione di memoria 36B alla locazione 399. Noi dovremo recuperare il primo della tabella per il carattere ASCII 2C, il successivo della tabella per il successivo nella sequenza di caratteri ASCII e così via. Questo è un caso tipico in cui è desiderabile usare l'indirizzamento indicizzato. Comunque qui sorge un altro problema: i caratteri ASCII sono numerati da 2C in avanti, invece che da "0" o da "1". La soluzione è abbastanza semplice ed appare di seguito:

```
TAX
LDA     TABLE-$2C, X
```

L'ASCII è trasferito nel registro indice X, cosicchè può essere usato come offset (fuori zero). Per tener conto del fatto che i caratteri sono numerati da 2C in avanti, si indica semplicemente l'inizio della tabella non all'indirizzo reale 36B, ma all'indirizzo della tabella meno 2C (esadecimale). Il codice binario può poi essere caricato nell'accumulatore con un semplice accesso indicizzato alla memoria (vedi Figura 4-32).

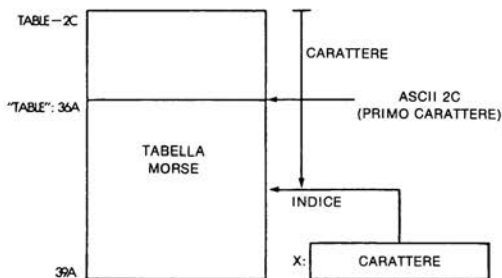


Figura 4-32: Impiego dell'Indirizzamento Indicizzato per Richiamare il Codice Morse

Il nostro codice Morse binario è ora nell'accumulatore. Ricordiamo che tale codice contiene un 1 iniziale, che è il bit START, seguito da 0 e 1 che rappresentano le linee ed i punti. Ogni bit non usato alla sinistra dello START, viene messo a 0. Il contenuto dell'accumulatore subirà, quindi, uno shift a sinistra fino a che non si trova il bit di START. poi i bit "reali", corrispondenti a linee o punti, vengono usati per generare suoni. Il programma è il seguente:

```
LDY     #$08      NUMERO DI BIT DA RUOTARE DA A
STY     COUNT
STARTB  ASL        A
DEC     COUNT
BCC     STARTB    SHIFT SU A FINO A TROVARE IL BIT START
STA     CHAR
```



NEXT	LDA	CHAR	
	ASL	A	SHIFT DEL CODICE MORSE (1 = LINEA, 0 = PUNTO)
	STA	CHAR	PUNTO = 1 INTERVALLO. MANCANZA A PUNTO
	LDY	#\$01	
	BCC	SEND	SE IL CARRY 0, PUNTO
	LDY	#\$03	ANCHE LINEA (TRE INTERVALLI)

Il registro indice Y potrà normalmente essere usato come un contatore, con lo scopo di fermare i successivi shift a sinistra di A, una volta che sono stati portati fuori 8 bit. In ogni caso, la routine SEND, che genererà il suono, richiede che il registro Y sia caricato con la durata del suono che deve essere prodotto. Si può quindi non usare il registro indice Y per portare fuori i bit. L'altra idea che viene in mente, è di riutilizzare il registro indice X, il quale ora è disponibile. Sfortunatamente, le nostre convenzioni in questo programma sono che la routine DELAY, impieghi il registro indice X. Dal momento che nessuno dei due registri è disponibile per un contatore, useremo una locazione di memoria. Tale, è la locazione COUNT. Un'importante considerazione è, che quando scriviamo il programma, dobbiamo avere ben codificato questa parte, prima di codificare la routine SEND o DELAY. In tal caso avremo probabilmente usato i registri indice X o Y, per memorizzare il numero di bit che devono subire lo shift nell'accumulatore. Più tardi potremmo scoprire di avere la necessità di usare quegli stessi registri nella routine SEND o DELAY. Questo accade quando si dà alla programmazione la massima importanza. Se si scopre che altre routine richiedono l'impiego di X e Y, bisogna tornare indietro alla codifica e cambiare le scelte nel programma che precede, usando una locazione di memoria indicata con COUNT invece di un registro. Il dimenticarsi quanto sopra è sfortunatamente un errore classico. In tal caso, infatti, si distruggerebbe accidentalmente il contenuto dei registri X e Y, provocando un grave errore di programmazione. Come regola si raccomanda, quindi, vivamente di scrivere esplicitamente nei commenti e all'inizio di ogni routine quali registri vengono cambiati o distrutti da quella routine. Le convenzioni per la comunicazione e il passaggio d'informazioni fra subroutine e segmenti di programma, dovranno essere completamente chiari prima della scrittura di una nuova routine.

	INDIRIZZO	SCRITTURA	LETTURA
TIMER 1	-- 04	T1L-L	T1C-L/ • azzerà il flag di int di T1
	-- 05	T1L-H + T1C-H + T1L-L → T1C-L • azzerà il flag di int di T1	T1C-H
	-- 06	T1L-L	T1L-L
	-- 07	T1L-H • azzerà il flag di int di T1	T1L-H
TIMER 2	-- 08	T2L-L	T2C-C • azzerà il flag di int di T2
	-- 09	T2C-H T2L-L → T2C-L • azzerà il flag di int di T2	T2C-H

Figura 4-33: Mappa di Memoria per il Timer 1

Gli zeri più a sinistra nell'accumulatore sono ignorati e il suo contenuto è spinto fuori fino a che non si trova il bit di START. Una volta che il bit START è stato trovato ogni bit spinto fuori sulla sinistra dell'accumulatore, rappresenta un "·" o una "—" in funzione del fatto che sia "0" o "1". Non appena il bit spinto fuori, con lo shift, dall'accumulatore è stato identificato si andrà alla locazione SEND per generare il tono corrispondente. Poiché il contenuto dell'accumulatore sarà cambiato dalle successive elaborazioni, esso deve essere conservato in memoria prima di andare alla locazione SEND. Questa è la funzione della seconda istruzione STA CHAR. Avendo conservato il contenuto dell'accumulatore nella locazione di memoria CHAR, il Registro Indice Y è caricato con la durata corrispondente al bit appena rilevato nell'accumulatore: un "1" se era un punto, un "3" se era una linea. La funzione di STA CHAR, seguita da LDA CHAR, che sembra inutile, è generata dal vincolo di voler rientrare in questo programma a "NEXT" con l'istruzione LDA CHAR.

### La Routine SEND

La routine SEND fa uso del timer 1 del 6522 per generare il tono di frequenza fissata. La mappa di memoria del timer è riportata in Figura 4-33. Il timer deve prima essere posto nel modo free-running. Questo lo si ottiene nel seguente modo:

```
SEND      LDA      #$C0
          STA      #A00B
```

Il valore C0 è inserito all'indirizzo A00B il quale è l'ACR o Registro di Controllo. Tale valore pone ad "1" i bit 6 e 7 come richiesto dal timer (vedere Figura 4-29 per i dettagli). Si inserisce poi il valore 0400 esadecimale agli indirizzi di memoria A006, A007:

```
LDA      #$00
STA      $A006
LDA      #$04
STA      $A007
```

Queste locazioni di memoria sono rispettivamente la parte bassa e la parte alta di T1L o latch. Ciò impone la frequenza del tono che si deve generare. 0400 in binario vale: 00000100 00000000 o 1024. Un semiperiodo del clock è approssimativamente  $N + 2$  o 1026. Il periodo vale quindi:

$$T = 2052 \text{ microsecondi}$$

$$\text{e la frequenza è } N = 1 \div T = \text{approssimativamente } 500 \text{ Hz}$$

Dobbiamo ora far partire il tono e fermarlo dopo il periodo specificato. L'inizio si ha con:

```
STA      $A005
```

Questa istruzione trasferisce i contenuti del latch nel contatore ed avvia la forma d'onda esterna. Abbiamo fatto notare che questo programma pone "on" anche "manualmente" PB0, cosicché un dispositivo esterno quale un trasmettitore, può essere attivato contemporaneamente alla generazione del tono nell'altoparlante. Questo lo si ottiene con:

```
LDA      #$01
STA      $A000
```

Si suppone che PB0 sia stato configurato come uscita, prima dell'esecuzione del programma.

La durata del tono è ottenuta dalla subroutine DELAY: JSR DELAY. La esamineremo di seguito. Una volta ottenuta la durata richiesta, il tono deve essere spento. Ciò si ottiene come segue:

LDA	#\$00	
STA	\$A00B	SPEGNI IL TONO
STA	\$A000	SPEGNI IL BIT D'USCITA (PB0)

Infine dobbiamo lasciare un'unità di silenzio fra due toni.

Questo è ottenuto con le istruzioni:

LDY	#\$01	
JSR	DELAY	1 - PERIODO DI RITARDO

Alla fine, dobbiamo decrementare il nostro contatore di bit, posto alla locazione di memoria COUNT, per controllare se nessun altro bit deve subire shift nell'accumulatore. Lo si ottiene con:

DEC	COUNT	SONO STATI FATTI 8 BIT?
BNE	NEXT	SE NO, TORNA INDIETRO

Una volta trasmesso un carattere completo, devono essere inserite due unità di ritardo, per separare questo carattere dal successivo. Allo scopo si ha:

FINISH	LDY	#\$02
	JSR	DELAY
EXIT	RTS	

### La Subroutine DELAY

Questa subroutine implementa un ritardo di: (contenuto del Registro Y) × (SPEED) × 0,001 secondi. Il ritardo sarà quindi calcolato come il prodotto di tre termini. Sarà qui usata una speciale tecnica di loop nidificati per ottenere una moltiplicazione classica. La routine è riportata di seguito:

DELAY	LDA	SPEED
D2	LDX	#\$FA
D1	DEX	
	BNE	D1
	SEC	
	SBC	#\$01
	BNE	D2
	DEY	
	BNE	DELAY
	RTS	

Il diagramma di flusso corrispondente compare in Figura 4-34.

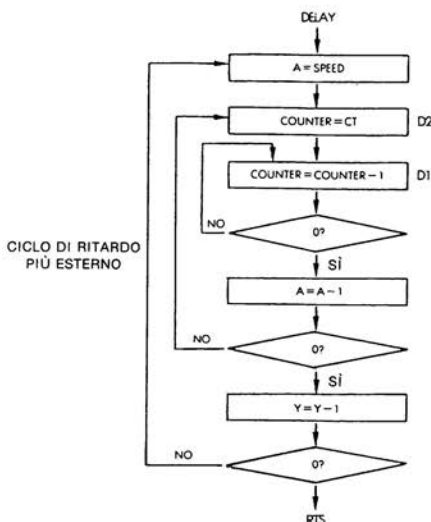


Figura 4-34: Diagramma di flusso per il Ritardo

Il primo loop di ritardo è quello corrispondente a D1. Calcoliamo la sua durata (fra parentesi è riportato il tempo di ogni istruzione):

(3)	LDA	SPEED	
(2)	LDX	#\$C6	C6 ESADEC = 198 DECIMALE
(2)	DEX		
(3)	BNE	D1	

La durata del ritardo introdotto da queste prime quattro istruzioni del programma è:  $3 + 2 + (2 + 3) \times 198 = 995$  microsecondi. Le successive due istruzioni sono:

(2)	SEC	
(2)	SBC	#\$01

La loro durata è di due microsecondi ognuna. Queste due istruzioni aggiungono, quindi, un ritardo di 4 microsecondi. Esse sono usate per sottrarre 1 al contenuto dell'accumulatore. Questo poiché entrambi i Registri Indice X e Y sono già stati usati, in questo programma, come contatori, cosicché l'accumulatore deve essere usato come terzo contatore. Sfortunatamente non c'è un'istruzione di decremento che operi direttamente sull'accumulatore, deve quindi essere usata un'istruzione di sottrazione. Il lettore ricorderà che il carry deve essere posto ad "1" prima di una sottrazione. Questa è la funzione dell'istruzione SEC prima della SBC. L'istruzione successiva è:

(2,3)	BNE	D2
-------	-----	----

Questo è un loop con un ritardo di un secondo. Ogni volta che il branch (diramazione) è verificato richiede 3 microsecondi e quando non si verifica richiede 2 microsecondi. Il ritardo totale dalla entrata al punto DELAY a questo punto della subroutine è quindi  $995 + 7 = 1002$  microsecondi = 1 millisecondo.

Si genererà un ritardo di 1 millisecondo ogni volta che si esegue il loop D2. Poiché D2 contiene il valore di SPEED, questi due loop realizzano un ritardo di  $SPEED \times 0,001$  secondi, che è quanto si voleva. Una volta raggiunto il ritardo totale di  $SPEED \times 0,001$  secondi, si esegue un ulteriore loop usando il registro Y:

```

DEY
BNE    DELAY
RTS

```

Tale loop finale moltiplica i ritardi precedenti per il valore contenuto nel Registro Y. A questo punto si è ottenuto il ritardo totale desiderato  $Y \times SPEED \times 0,001$  secondi e si esegue un return (RTS).

*Impiego del programma.* Allo scopo di usare questo programma, si suggerisce di scegliere una bassa velocità di trasmissione iniziale, a meno che non si abbia familiarità con il codice Morse e di generare un solo carattere alla volta. Una volta constatato che il programma lavora correttamente, si potrà scrivere una breve subroutine che fornirà i caratteri al programma Morse. Si potrà poi verificare che la trasmissione Morse proceda correttamente per ogni stringa di caratteri.

**ESERCIZIO 4-2:** Scrivere una subroutine che fornisca al programma una stringa di N caratteri contenuti in una tabella con inizio all'indirizzo TABLE.

**ESERCIZIO 4-3:** Leggere la tastiera e generare i corrispondenti segnali Morse.

## OROLOGIO

Svilupperemo qui una routine per la misura del tempo quotidiano (Time of Day clock) che conserva in tre locazioni di memoria il tempo in ore, minuti e secondi. Se si desidera, questo programma può velocemente essere esteso alla memorizzazione di frazioni di secondi, od ogni altra unità di tempo desiderata. La mappa di memoria di questo programma è riportata in Figura 4-35. Come al solito, le locazioni di memoria in Pagina 0 (zero) sono riservate alle variabili. Le ore, i minuti, i secondi, sono memorizzati rispettivamente alle locazioni di memoria 00F4 (esadecimale), 00F5, 00F6. Si usa un'ulteriore locazione: 00F7 che contiene la variabile COUNT.

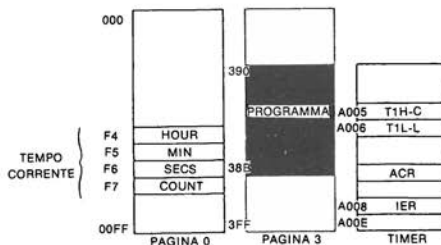


Figura 4-35: Mappa di Memoria dell'Ora

Per far partire l'orologio, il programma deve essere scritto, poi bisogna inserire nelle locazioni SECS, MIN, HOUR il tempo delle correnti 24 ore più un minuto.

Bisogna poi inserire A7 nella locazione A67E (per SYM) e 03 nella locazione A67F. Questo è un vettore di interrupt la cui funzione sarà spiegata più avanti. Infine bisogna inserire "GO 0390"; poi all'ora esatta che è stata inserita in SECS, MIN, HOUR premere "CR".

L'ora esatta sarà, da quell'istante, conservata in SECS, MIN, HOUR.

La variabile COUNT memorizza unità di 20-esimi di secondo. Essa è posta inizialmente al valore 20, poi è decrementata ogni 20-esimo di secondo. Il segnale per il decremento è un interrupt hardware generato da un timer contenuto nel 6522. Il diagramma di flusso del programma appare in Figura 4-36. La prima fase è quella di inizializzazione, dove il timer è caricato con l'opportuno valore del contatore, per generare un interrupt dopo 50 millisecondi (1/20-esimo di secondo). La variabile COUNT è inizializzata al valore 20 e il timer è fatto partire.

Quando il timer è arrivato a fine conteggio si è raggiunto 1/20-esimo di secondo e viene generato un interrupt. Alla ricezione di un interrupt, il microprocessore conserva i propri registri, ricarica il registro del contatore del timer, con la costante opportuna, per la generazione di un altro interrupt 50 millisecondi dopo, quindi fa ripartire il timer. La locazione di memoria COUNT sarà decrementata, dal momento che si è raggiunto 1/20-esimo di secondo. Il valore di questa locazione sarà confrontato con il valore "0". Se non è "0" si ha l'uscita da questa routine. Quando COUNT raggiunge "0" essa è riportata al valore "20" e la locazione di memoria SECS (il numero di secondi) è incrementato di 1.

Ogni volta che SECS è incrementato di 1, il suo valore viene confrontato con "60". Se si è raggiunto il valore "60", SECS deve essere posto a "0" e MIN (il numero di minuti) deve essere incrementato. Allo stesso modo MIN deve essere confrontato con "60". Se MIN ha raggiunto "60" esso deve essere posto a "0" e si deve incrementare il numero delle ore. Se il numero di ore raggiunge 24 esso è riportato a "0". Si ha poi l'uscita da questa routine. Il programma rimarrà inattivo fino alla ricezione dell'interrupt successivo. Allo scopo di mettere su display il contenuto di questo time-of-day-clock, l'utilizzatore deve semplicemente esaminare il contenuto delle locazioni di memoria F4, F5 e F6. Si può anche scrivere una routine per porre automaticamente su display il valore di quelle locazioni di memoria.

Il programma compare in Figura 4-37 e si spiega da solo.

Il primo tratto di programma è relativo alla inizializzazione INIT, che pone la variabile COUNT al valore "20" decimale = "14" esadecimale. Esso carica anche il timer con l'opportuno valore per generare un ritardo di 50 millisecondi. In Figura 4-35 è riportata la mappa di memoria per il timer. Si impiega il timer 1 del 6522.

La tabella che indica i bit per il condizionamento di questo dispositivo, appare in Figura 4-25 e 4-29. Il suddetto timer può essere usato sia in modo one-shot che in quello free-running. In one-shot, sarà generato un solo interrupt (e possibilmente un impulso d'uscita su PB7), ogni volta che il contatore interno del timer, arriva a 0 (zero). In free-running, il contatore sarà automaticamente ricaricato dal latch interno e saranno generati interrupt continuamente (e possibilmente un impulso su PB7). Poiché il pin d'uscita PB7 non è usato in questa applicazione, il bit 7 di ACR (registro di controllo ausiliario), sarà posto a "0". C'è quindi da scegliere fra il modo one-shot e il modo free-running. Nel one-shot il contatore deve essere ricaricato ogni volta che si genera interrupt. Nel free-running il contatore del timer sarà ricaricato automaticamente dal suo latch. Ad ogni modo, il flag di interrupt deve essere azzerato direttamente scrivendo in TIC-H oppure, modificando il flag stesso. Le due soluzioni sono essenzialmente identiche in termini di sforzi di programmazione. Il modo free-running può fornire una più accurata misura del tempo, dal momento che il timer passa continuamente e automaticamente dal valore "0" al valore corrispondente al ritardo di 50 millisecondi. Poiché nel programma Morse si è usato il modo free-running, qui impiegheremo in one-shot. Si suggerisce al lettore di sperimentare, per esercizio, l'impiego del metodo alternativo. L'impiego nel modo one-shot è specificato ponendo

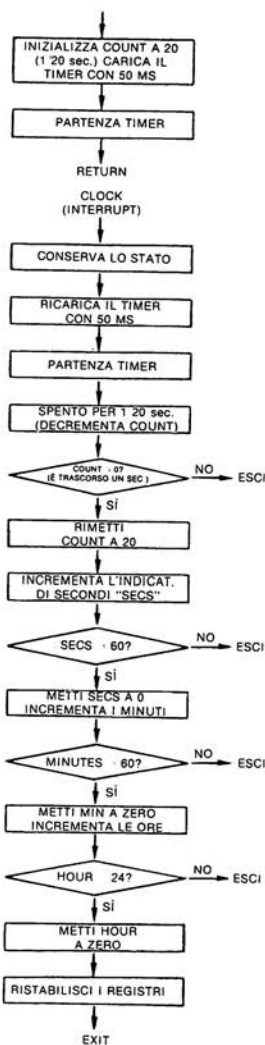


Figura 4-36: Orologio  
per l'Ora del Giorno

do il bit ACR6 a "0". Tutti gli altri bit del registro ACR, che qui non sono usati, vengono posti a "0". I bit 7 e 8 di ACR vengono posti a "0" per indicare il modo one-shot con PB7 disabilitato.

Inoltre, bisogna condizionare opportunamente il registro dei flag di interrupt. Quando viene letto, tale registro è analizzato come un Registro Dei Flag di Interrupt, IFR. Quando si scrive nel suddetto registro, esso viene chiamato Registro Enable Interrupt, IER. Allo scopo di posizionare i bit specifici di IER, bisogna mettere ad 1 il bit 7 di IER. Per ogni "1" indicato nelle diverse locazioni del registro, da 0 fino a 6, sarà scritto un 1 nel registro, abilitando la relativa condizione. Uno "0" in ogni posizione di bit, non cancella la posizione di bit specificata nel registro IER, ma lascia invariato il contenuto. L'azzeramento è ottenuto imponendo "0" nel bit di posizione 7, poi inserendo "1" per ogni posizione di bit che deve essere cancellata. Nel nostro caso, si vuole abilitare semplicemente un interrupt dal timer T1. Scriveremo quindi il valore "11000000", o "C0" in esadecimale, nella locazione di memoria corrispondente a IER (Vedere Capitolo 2 per i dettagli).

```

LINE #LOC CODE LINE
;CARICA PRELIM A7 NELLA LOC A67E E 03 IN A07F
0002 0000 ;QUESTA È UNA ROUTINE DI OROLOGIO IN TEMPO REALE
0003 0000 ;CHE CONSERVA IL TEMPO CORRENTE ALLE LOC SEC (00F8), MIN
0004 0000 ;(005F) ED HOUR (00F4). ESSA È INTERROTTA
0005 0000 ;DA FINE TEMPO O INTERRUPT TIMER, CHE
0006 0000 ;CAUSA UN INTERRUPT E SALTA ALLA ROUTINE
0007 0000 ;DELL'OROLOGIO 20 VOLTE AL SECONDO LA ROUTINE DELL'OROLOGIO
0008 0000 ;ED IL TIMER DELL'INTERVALLO DEVONO PRIMA ESSERE INIZ
0009 0000 ;QUESTO VIENE ESEGUITO DAL COD 'INIT' E SI DEVE SALTARE
0010 0000 ;ALL'INIZIO PER INIZIALIZZ. INTRODURRE IL TEMPO ATTUALE
0011 0000 ;LA ROUTINE PARTIRÀ A SEC. MIN ED
;HOUR INVIANDO IL COMANDO 'GO 0380 CR'
;NON OCCORRE FARE ALTRO
0012 0000 COUNT - 0007 ;CONTATORE PER VENTESIMI DI SECONDO
0013 0000 SECS - 000F ;TEMPO CORRENTE
0014 0000 MIN - 0005
0015 0000 HOUR - 0004
0016 0000 ACR - 0A0B ;REGISTRO TIMER MODE
0017 0000 T'LL - 0A06 ;CONSTANTE DEL TIMER DI BASSO ORDINE
0018 0000 T'HC - 0A05 ;CONSTANTE DEL TIMER DI ORDINE ELEVATO
0019 0000 ;- B0390
0020 0390 A9 14 INIT LDA #014 ;POSIZIONATI AI PRIMI ENTI
0021 0391 85 F7 STA COUNT ;CONTEGGI
0022 0394 8D 0B A0 STA ACR ;PONI A 0 IN ACR 1
;BIT 7 ED 8
0023 0397 A9 C0 LDA #0C0 ;PONI AD 1 I BIT 8 E 7
0024 0399 8D 0E A0 STA 0A0E ;NEL REGISTRO DI ABILITAZIONE
;INTERRUPT (PER ABILITARE
;GLI INTERRUPT DAL TIMER 1)
0025 039C A9 50 LDA #050 ;MEMORIZZA C350 NEL TIMER
0026 039E 8D 06 A0 STA T'LL ; (CONSTANTE DI RITARDO
0027 03A1 A9 C3 LDA #0C3 ; PER 50 MS)
0028 03A3 8D 05 A0 STA T'HC ;QUESTO A' VIA IL TIMER
0029 03A6 60 RTS ;RITORNA AL MONITOR
0030 03A7 08 CLOCK PhP ;SALVA LO STATO
0031 03A8 48 PHA
0032 03A9 F8 SEO
0033 03AA A9 50 LDA #050 ;MEMORIZZA C350 NEL TIMER
0034 03AC 8D 06 A0 STA T'LL ; (CONSTANTE DI RITARDO
0035 03AF A9 C3 LDA #0C3 ; PER 50 MS)
0036 03B1 8D 05 A0 STA T'HC ;QUESTO AVVIA IL TIMER
0037 03B4 C6 F7 DEC COUNT ;DECREMENTA IL CONTEGGIO
;DI VENTI
0038 03B6 D0 31 BNE EXIT ;ESCI SE NON SI È
;ANCORA CONTATO FINO A VENTI
0039 03B8 A9 14 LDA #014 ;ALTRIMENTI RIPRISTINA CONTEGGIO

```

Figura 4-37: Il Programma per l'Ora del Giorno (Listing completo in Appendice C)



```

0040 03BA 85 F7      STA COUNT      ;UN INTERO SECONDO E TRASCOF. O
0041 03BC A9 01      LDA #501
0042 03BE 18         CLC
0043 03BF 85 F6      ADC SECS      ;AGGIUNGI 1 A SEC
0044 03C1 85 F6      STA SECS
0045 03C3 C9 60      CMP #560      ;EDI SE SI E ARRIVATI A 60 SECONDI
0046 03C5 D0 22      BNE EXIT      ;SE NO. ESCI
0047 03C7 A9 00      LDA #500      ;ALTRIMENTI PONI SECONDI A 0
0048 03C9 85 F6      STA SECS
0049 03CB A9 01      LDA #501
0050 03CD 18         CLC
0051 03CE 85 F5      ADC MIN      ;E SOMMA 1 AI MINUTI
0052 03D0 85 F5      STA MIN
0053 03D2 C9 60      CMP #560      ;EDI SE SI E RAGGIUNTO 60 MINUTI
0054 03D4 D0 13      BNE EXIT      ;SE NO. ESCI
0055 03D6 A9 00      LDA #500
0056 03D8 85 F5      STA MIN      ;ALTRIMENTI PONI 1 MINUTI A 0
0057 03DA A9 01      LDA #501
0058 03DC 18         CLC
0059 03DE 85 F4      ADC HOUR      ;E SOMMA 1 AD HOUR
0060 03DF 85 F4      STA HOUR
0061 03E1 C9 24      CMP #524      ;EDI SE SI SONO RAGGIUNTE 24 ORE
0062 03E3 D0 04      BNE EXIT      ;SE NO. ESCI
0063 03E5 A9 00      LDA #500
0064 03E7 85 F4      STA HOUR      ;ALTRIMENTI PONI HOUR A 0
0065 03E9 68         EXIT PLA      ;RIMEMORIZZA LO STATO
0066 03EA 28         PLP
0067 03EB 40         RTI

```

ERRORS : 0000 < 0000 >  
 SYMBOL TABLE  
 SYMBOL ALUE  
 A:IR A00B CLOCK 03A7 COUNT 00F7 EXIT  
 HOUR 00F4 INIT 0390 MIN 00F5 PLS  
 SECS 00F6 T1HC A005 T3LL A006  
 END OF A'SEMBLY

Figura 4-37: Il Programma per l'Ora del Giorno (continua)

Infine, dobbiamo caricare nel timer, la costante adeguata per ottenere un ritardo che genererà un interrupt da 50 millisecondi. Si carica quindi il valore C350 esadecimale (= 50000 decimale) nel contatore. Si vedrà, nella routine INIT, che si carica prima la parte bassa del latch, poi la parte alta del contatore. Caricando la parte alta del contatore, si ottiene automaticamente il trasferimento della parte più bassa del latch nella parte più bassa del contatore e contemporaneamente la partenza del timer.

La subroutine INIT è riportata di seguito:

```

COUNT = $00F7      1,20 DI SECONDO
SECS    = $00F6
MIN     = $00F5
HOUR    = $00F4
ACR     = $A00B      REGISTRO DI MODO DEL TIMER
T3LL    = $A006      PARTE BASSA DEL TIMER CT
T1CH    = $A005      PARTE ALTA DEL TIMER CT
INIT     LDA #14      PRIMI VENTI GIRI
        STA COUNT
        STA ACR       BIT 7 E 8 DI ACR BASSI
        LDA #0C0      BIT 7 E 8 ALTI
        STA $A00E     NEL REGISTRO ENABLE INTERRUPT

```

LDA	=50	MEMORIZZA C350 NEL TIMER
STA	T1LL	(CT PER 50 MS)
LDA	#C3	
STA	TICH	PARTENZA TIMER
RTS		

L'inizializzazione è stata ora completata e si esegue il programma principale dalla locazione CLOCK. Si noti che tutte le addizioni all'interno della routine CLOCK, sono eseguite in decimale. Questo perché il flag decimale è posto ad "1" con l'istruzione SED. In tal modo, quando si manda su display il contenuto delle locazioni di memoria, si visualizzerà un digit per LED, nella solita maniera decimale invece che in forma esadecimale.

Dopo l'esecuzione della subroutine INIT, si ha il ritorno al monitor. Se non si piglia nessun tasto della tastiera, non avverrà nulla fino a che non si genererà un interrupt di time-out. Al rilevamento dell'interrupt si avrà un salto automatico nell'orologio. Quando nel 6502, avviene un interrupt, esso salta automaticamente alla locazione di memoria FFFE, FFFF dove trova il vettore di interrupt, cioè l'indirizzo successivo da inserire nel program counter. Nel SYM, l'utilizzatore carica all'inizio, le locazioni di memoria A67E e A67F con il vettore di interrupt desiderato. Il monitor del SYM, che è attivo tutte le volte che non si sta eseguendo il programma dell'utilizzatore, duplica automaticamente il contenuto delle locazioni A600 fino A67F, riportandolo anche agli indirizzi da FF80 a FFFF. In questo modo il contenuto di A67E e A67F, viene copiato automaticamente dal monitor del SYM agli indirizzi di memoria FFFE, FFFF. Nell'istante in cui si ha interrupt, si salterà a FFFE, FFFF e lì si troverà un contenuto a 16 bit da inserire nel program counter.

CLOCK è la routine di interrupt nella quale si entra ogni volta che si riceve un interrupt. Essa salva il contenuto del registro P (registro di stato) ed A (accumulatore). Non è necessario salvare gli altri registri come lo è per questi.

Viene poi ricaricato il contatore del timer con il valore C350 esadecimale = 50000 decimale e il timer riparte di nuovo. Caricando il contatore si azzera automaticamente il precedente interrupt.

La routine poi, in seguito, controlla se la variabile COUNT ha raggiunto il valore "20", se SECS ha raggiunto il valore "60", se MIN ha raggiunto il valore "60" o se la variabile HOUR ha raggiunto il valore "24". Se qualcuna delle variabili ha eguagliato il suo valore limite, essa è posta a "0", come indicato nel flow-chart di Figura 4-36, o il programma di Figura 4-37.

Infine si esce dalla routine ripristinando i due registri che si erano salvati, A e P ed eseguendo una RTI (Return From Interrupt).

## UN PROGRAMMA DI CONTROLLO DOMESTICO

Un programma per home control generalizzato, metterà su di un monitor l'ora esatta, lo stato di un sistema di allarme e farà diverse azioni in funzione dell'ora del giorno, o in seguito al rilevamento di una condizione di allarme. Sarà usato qui il programma di misura dell'ora quotidiana che è stato sviluppato in precedenza, mostrando su display l'ora del giorno, poi in funzione di questa saranno compiute particolari funzioni chiudendo uno o più relé. Il programma è riportato in Figura 4-38. Il registro di direzione dati della Porta B è posto al valore esadecimale 0F, con lo scopo di abilitare i quattro bit inferiori come uscite (per i relé). Chiaramente, solo quei bit attualmente collegati ai relé dovranno essere indicati come uscite. Gli altri dovranno rimanere ingressi. Come al solito, per precauzione, è inclusa nel programma una istruzione esplicita per aprire i relé. Questo lo si ottiene inserendo il valore esadecimale 00, alla locazione di memoria di IORB (indirizzo AC00).

In questo programma sono usate due routine del monitor del SYM per facilitare l'uscita. L'accumulatore è caricato con il contenuto della locazione di memoria HOUR che contiene l'ora del giorno espressa in ore (vedi la routine di calcolo dell'ora quotidiana), poi si esegue una chiamata alla subroutine OUTBYT che permette la visualizzazione di HOUR sul display. Allo stesso modo vengono visualizzati i minuti, caricando l'accumulatore con il contenuto della locazione di memoria MIN e chiamando OUTBYT.



La routine OUTBYT è contenuta alla locazione di memoria 82FA del monitor e visualizza il contenuto di A su due digit esadecimali. Successivamente, si usa la routine SCAND (all'indirizzo di memoria 8906), per eseguire una scansione del display. Una volta visualizzata l'ora, si eseguirà un'opportuna istruzione di salto, se si verificano alcune condizioni. Poiché tali condizioni saranno diverse nelle varie applicazioni, nel programma si è lasciato uno spazio vuoto che potrà essere riempito dal lettore. Per esercizio si suggerisce di attivare 2 o 3 volte i relé, separatamente, per pochi minuti. Il rumore fatto dai relé quando si chiudono indica che il programma è eseguito correttamente. Questo dovrà essere fatto prima di collegare qualsiasi dispositivo ai relé.

## COMBINATORE TELEFONICO

Svilupperemo un programma capace di comporre un numero una volta che questo è stato caricato in memoria. Con un telefono tradizionale (combinatore rotante), si inviano semplicemente impulsi sulla linea. Questo ora può essere fatto semplicemente e noi svilupperemo qui un programma in grado di generare le frequenze usate negli Stati Uniti per telefoni a tastiera.

La tabella delle frequenze telefoniche appare in Figura 4-39. Ogni digit permette la generazione di due toni. Le diverse frequenze sono state scelte accuratamente dalla compagnia telefonica con lo scopo di evitare la possibilità di armoniche spurie, e per occupare la più piccola banda possibile. Essi variano nel range da 697 Hertz a 1477 Hertz, come indicato nell'illustrazione.

Il nostro programma genererà simultaneamente due toni, che saranno introdotti nello stesso altoparlante. Le frequenze dovranno essere precise per poter essere riconosciute dal circuito switching del telefono.

			TONI BASSI
1	2	3	697
4	5	6	770
7	8	9	852
*	0	#	941
TONI ALTI			
1	2	3	1209
4	5	6	1336
7	8	9	1477

Figura 4-39: Le Frequenze Telefoniche

Questo risultato può essere ottenuto impiegando due timer. Useremo qui il timer A ed il timer B della scheda del microprocessore. Ogni timer genererà una frequenza e l'uscita di entrambi i timer sarà inviata all'altoparlante. Per avere risultati più affidabili, si raccomanda di fare uso di un amplificatore operazionale per l'altoparlante. Ad ogni modo il programma rimarrà invariato. Il diagramma di flusso del programma appare in Figura 4-40. Il numero di digit per il numero telefonico è irrilevante. Questo programma manipolerà un numero telefonico di lun-

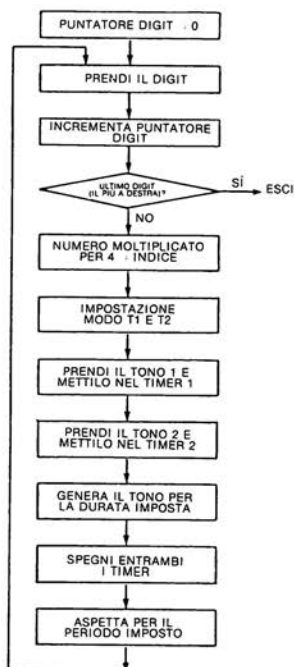


Figura 4-40: Diagramma di flusso per Combinatore Telefonico

ghezza qualsiasi. Il primo digit da "comporre" è ottenuto dalla memoria. Una tabella delle equivalenze, che indica i periodi per i due toni da generare per ogni digit, è conservata in memoria. Più precisamente tale tabella indica il *semiperiodo* e dal momento che i due toni sono associati ad ogni digit, questa tabella userà quattro byte per ogni digit. Il valore del digit deve quindi essere moltiplicato per quattro per poter essere usato come indice di questa tabella.

Si otterranno due valori di tabella e si caricheranno rispettivamente nel timer A e B che saranno avviati. I due toni saranno poi generati automaticamente per una specifica durata (diciamo mezzo secondo o un secondo). Sarà poi inserito un intervallo di silenzio e si prenderà dalla memoria il digit successivo. Il procedimento sarà ripetuto fino a che tutti i digit sono stati composti. Il flow-chart è semplice. Esaminiamo ora il programma. Il programma completo è mostrato in Figura 4-41.

LINE	LOC	CODE	LINE
0002	0000		:QUESTO È UN PROGRAMMA CHE COMPONE
0003	0000		:NUMERI TELEFONICI PRE-MEMORIZZATI, ESSO PRODUCE UN'USCITA
0004	0000		:A DUE TONI SU UN ALTOPARLANTE CONNESSO IN CONFIGURAZIONE 2
0005	0000		: (2 TONI - VEDI ALTOPARLANTE). QUESTI TONI RIPRODURRANNO
0006	0000		:IL TOCCO TELEFONICO PONENDO L'ALTOPARLANTE IN
0007	0000		:CORRISPONDENZA DELLA CORNETTA LATO BOCCA
0008	0000		:PER IMPIEGARE IL PROGRAMMA CARICA IL NUMERO (1)
0009	0000		:TELEFONICO DOVUNQUE IN MEMORIA, UNA CIFRA PER BYTE
0010	0000		:TERMINANDO CON OF (ESADEC.). PER ES. IL NUM.
0011	0000		:555-1212 RISULTEREBBE IN MEMORIA 05 05 01 02 01 02 0F (TUTTI ESADEC.)
0012	0000		:CARICANDO L'INDIRIZZO DEL NUMERO SI INTRODUCA
0013	0000		:PRIMA IL BYTE BASSO, NELLE LOC. 00C0 E 00C1.
0014	0000		:INOLTRE SI ENTRA A QUESTA SUB.
			:DA MONITOR O JSR AD ESSA DA UN ALTRO PROGRAMMA
0015	0000		NUMPTR = \$00C0 :QUESTO PUNTA ALL'INDIRIZZO
			:DEL NUMERO TELEFONICO
0016	0000	ONDEL = \$40	:QUESTA È LA COSTANTE
			:DI RITARDO PER IL TEMPO
0017	0000	OFFDEL = \$20	:COSTANTE DI RITARDO PER IL TEMPO
			:QUANDO I TONI SONO 0
0018	0000	DELCON = \$FF	:COSTANTE DI RITARDO
			:PER SCOPI GENERALI
0019	0000	ACR1 = \$A0B8	:QUESTI SONO I REGISTER
			:TIMER MODE (TIMER 1)
0020	0000	ACR2 = \$AC08	: (TIMER 2)
0021	0000	T1CH = \$AC05	:QUESTO È IL CONTATORE DEL TIMER 1
			: (BYTE ELEVATO)
0022	0000	T1LH = \$A007	:LATCH DEL TIMER 1 (BYTE ELEVATO)
0023	0000	T1LL = \$A004	: (BYTE BASSO)
0024	0000	T2CH = \$AC05	:PER TIMER 2 ANALOGAMENTE A TIMER 1
0025	0000	T2LH = \$AC07	
0026	0000	T2LL = \$A004	
0027	0000		: ~ \$0300
0028	0300	PHONE LDY #300	:INDICE PER LE CIFRE
			:DEL NUMERO TELEFONICO
0029	0302	B1 C0 DIGIT LDA (NUMPTR,Y)	:ACCETTA CIFRA
0030	0304	C8 INY	
0031	0305	C9 OF CMP #50F	:VEDI SE FINE
			:DEL NUMERO TELEFONICO
0032	0307	D0 01 BNE NOEND	
0033	0309	60 RTS	:RITORNO SE SI (AL
			:MONITOR O
			:PROGRAMMA CHIAMANTE)
0034	030A	0A EA EA NOEND ASL A	:MULTIPLICA IL NUMERO PER
			:QUATTRO ALLA TABELLA INDICE
0035	030D	0A EA EA ASL A	: (OGNI INGRESSO ALLA TABELLA
			: È DI QUATTRO BYTE)
0036	0310	14 TAX	:X = INDICE DELLA TABELLA
0037	0311	A9 C0 LDA #5C0	
0038	0313	8D 08 A0 STA ACR1	:PONI IL MODO DEL TIMER
			:A FREE RUNNING SU ENTRAMBI I TIMER
0039	0318	8D 08 AC STA ACR2	
0040	0319	8D 5D 03 LDA TABLE,X	:ACCETTA IL BYTE DI BASSO
			:ORDINE DEL PRIMO TONO
0041	031C	8D 04 A0 STA T1LL	:MEMORIZZA IL TIMER 1
0042	031F	E8 INX	
0043	0320	8D 5D 03 LDA TABLE,X	:ACCETTA IL BYTE DI ORDINE
			:ELEVATO DEL PRIMO TONO
0044	0323	8D 07 A0 STA T1LH	:MEMORIZZA IL TIMER 1
0045	0326	8D 05 A0 STA T1CH	:QUESTO AVVIA
			:IL TIMER 1
0046	0329	E8 INX	
0047	032A	8D 5D 03 LDA TABLE,X	:ACCETTA IL BYTE DI BASSO
			:ORDINE DEL SECONDO TONO
0048	032D	8D 04 AC STA T2LL	:MEMORIZZA NEL TIMER 2
0049	0330	E8 INX	
0050	0331	8D 5D 03 LDA TABLE,X	:ACCETTA IL BYTE DI ORDINE
			:ELEVATO DEL SECONDO TONO

Figura 4-41: Programma per Combinatore Telefonico (Listing completo in Appendice C)

```

0051 0334 8D 07 AC STA T2LH ;MEMORIZZALO NEL TIMER 2
0052 0337 8D 05 AC STA T2CH ;QUESTO AVVIA
                                ;IL TIMER 2
0053 033A A2 40 LDX #ONDEL ;ACCETTA LA COSTANTE DI
                                ;RITARDO DI TONO ACCESO
0054 033C 20 55 03 ON JSR DELAY ;RITARDO MENTRE IL TONO È ACCESO
0055 033F CA DEX ;
0056 0340 D0 FA BNE ON ;
0057 0342 A9 00 LDA #800 ;
0058 0344 8D 0B A0 STA ACR1 ;DISABILITA ENTRAMBI I TIMER
0059 0347 8D 0B AC STA ACR2 ;
0060 034A A2 20 LDX #OFFDEL ;ACCETTA LA COSTANTE DI RITARDO
                                ;DI TONO SPENTO
0061 034C 20 55 03 OFF JSR DELAY ;RITARDO MENTRE IL TONO È SPENTO
0062 034F CA DEX ;
0063 0350 D0 FA BNE OFF ;
0064 0352 4C 02 03 JMP DIGIT ;RITORNA INDIETRO PER LA CIFRA
                                ;SUCCESSIVA DEL NUMERO TELEFONICO

0065 0355 ;
0066 0355 ;QUESTA È UNA SEMPLICE ROUTINE DI RITARDO
                                ;PER L'ACCENSIONE E SPEGNIMENTO DEL TONO

0067 0355 ;
0068 0355 A9 FF DELAY LDA #DELCON ;ACCETTA LA COSTANTE
0069 0357 38 WAIT SEC ;DI RITARDO DI QUESTA LUNGHEZZA
0070 0358 E9 01 SBC #501 ;
0071 035A D0 FB BNE WAIT ;
0072 035C 80 RTS ;
0073 035D ;
0074 035D ;QUESTA È UNA TABELLA DELLE COSTANTI PER LE FREQUENZE
0075 035D ;DEL TONO PER OGNI CIFRA TELEFONICA LE COSTANTI
0076 035D ;SONO LUNGHE 2 BYTE. PRIMA QUELLO BASSO
0077 035D ;
0078 035D 13 TABLE ;BYTES$13, $02, $76, $01: DUE TONI PER '0'
0078 035E 02
0078 035F 76
0078 0360 01
0079 0361 CD ;BYTES$CD, $02, $98, $01: DUE TONI PER '1'
0079 0362 02
0079 0363 9E
0079 0364 01
0080 0365 CD ;BYTES$CD, $02, $76, $01: '2'
0080 0366 02
0080 0367 78
0080 0368 01
0081 0369 CD ;BYTES$CD, $02, $53, $01: '3'
0081 036A 02
0081 036B 53
0081 036C 01
0082 036B 89 ;BYTES$89, $02, $9E, $01: '4'
0082 036E 02
0082 036F 9E
0082 0370 01
0083 0371 89 ;BYTES$89, $02, $76, $01: '5'
0083 0372 02
0083 0373 76
0083 0374 01
0084 0375 89 ;BYTES$89, $02, $53, $01: '6'
0084 0376 02
0084 0377 53
0084 0378 01
0085 0379 4B ;BYTES$4B, $02, $9E, $01: '7'
0085 037A 02
0085 037B 9E
0085 037C 01
0086 037D 4B ;BYTES$4B, $02, $76, $01: '8'
0086 037E 02
0086 037F 76
0086 0380 01
0087 0381 4B ;BYTES$4B, $02, $53, $01: '9'

```

Figura 4-41: Programma per Combinatore Telefonico (continua)

```

0087 0382 02
0087 0383 03
0087 0384 01
0088 0385      FINE
ERRORS 0000 < 0000 >
SYMBOL TABLE
ACR1  A00B      ACR2  AC0B      DELAY  0355      DELCON  00FF
DIGIT  0302      NOEND 030A      NUMPTR  00C0      OFF  034C
OFFDEL 0020      ON  033C      ONDEL  0040      PHONE  0300
T1CH  A005      T1LH  A007      T1LL  A004      T2CH  AC05
T2LH  AC07      T2LL  AC04      TABLE 035D      WAIT  0357
END OFF + _SEMBLY

```

Figura 4-41: Programma per Combinatore Telefonico (continua)

Il Registro Y è usato come puntatore del digit corrente del numero telefonico che deve essere composto. Esso è innanzi tutto inizializzato a 0:

```
PHONE    LDY    #$00
```

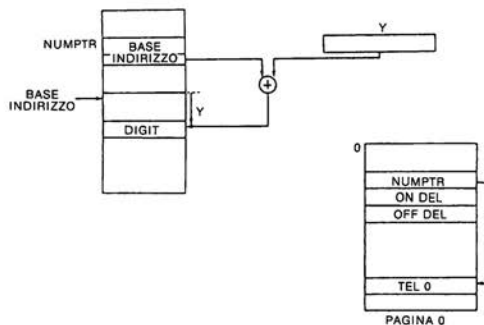


Figura 4-42: Combinatore Telefonico: Accesso Indiretto Indicizzato (in alto) e Mappa di Memoria (in basso)

Successivamente il digit è ottenuto impiegando una tecnica di indirizzamento indicizzato indiretto (vedi Figura 4-42). Si suppone che il numero telefonico completo sia stato memorizzato sequenzialmente partendo dall'indirizzo "NUMPTR", e termini con il valore "0F", che indica la fine del numero telefonico.

```
DIGIT    LDA    (NUMPTR),Y    ACCETTA DIGIT
```

Il Registro indice Y è poi incrementato, cosicché punterà il digit successivo per il prossimo giro. Si controlla se si è trovato l'ultimo digit ("0F") e se così è, il programma termina:

```

INY
CMP      #$0F
BNE      NOEND
RTS

```



Supporremo di non avere ancora raggiunto l'ultimo digit del numero telefonico e quindi procederemo. Il valore del digit deve essere moltiplicato per quattro, dal momento che, come abbiamo già detto, la tabella delle equivalenze fra i digit e i semiperiodi contiene quattro byte per ingresso. La moltiplicazione per quattro si eseguirà con due successivi shift a sinistra. Il risultato sarà poi memorizzato nel registro X cosicché possa essere usato come indice:

```
NOEND    ASL      A
          ASL      A
          TAX
```

Successivamente entrambi i timer A e B, sono posti nel modo free running:

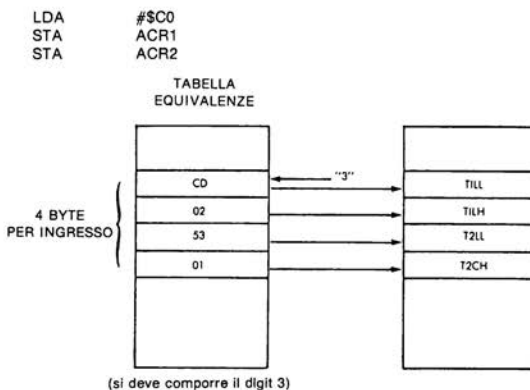


Figura 4-43: Caricamento del Timer

Essi sono poi caricati ognuno con il semiperiodo ricavato dalla tabella delle equivalenze (vedi Figura 4-43):

```

LDA      TABLE,X
STA      T1LL
INX
LDA      TABLE,X
STA      T1LH
STA      T1CH
INX
LDA      TABLE,X
STA      T2LL
INX
LDA      TABLE,X
STA      T2LH
STA      T2CH
```

Una volta che entrambi i timer sono stati azionati, si deve semplicemente generare il tono per un determinato periodo di tempo. Tale durata è qui indicata dal valore ONDEL. Il ritardo richiesto è ottenuto con la subroutine DELAY ed un loop secondario "ON".

```

ON      LDX      #ONDEL
        JSR      DELAY
        DEX
        BNE      ON

```

Infine, una volta che si è generato il tono per il periodo giusto di tempo, entrambi i timer vengono semplicemente disattivati:

```

        LDA      #$00
        STA      ACR1
        STA      ACR2

```

poi si genera un ritardo di silenzio:

```

OFF     LDX      #OFFDEL
        JSR      DELAY
        DEX
        BNE      OFF

```

e il programma ritorna all'inizio per generare i toni corrispondenti al digit successivo:

```

        JMP      DIGIT

```

La routine DELAY è del tipo classico:

```

DELAY   LDA      #DELCON
WAIT    SEC
        SBC      #$01
        BNE      WAIT
        RTS

```

La tabella delle equivalenze che indica il semiperiodo equivalente, per i toni che devono essere generati, appare alla fine del programma in Figura 4-41.

Calcoliamo i semiperiodi corrispondenti a ognuna delle frequenze che sono state generate. Devono essere generate sette frequenze: 697, 770, 852, 941, 1209, 1336, 1477.

Ad esempio, per una frequenza  $N = 697 \text{ Hz}$ , il corrispondente periodo è  $1/N = 1434,7$  microsecondi. Il semiperiodo è quindi 717 microsecondi o 02CD esadecimale.

FREQUENZA DESIDERATA	SEMI- PERIODO	$N = \text{SEMI-}$ $\text{PERIODO} - 1,7$	ESADEC. per Eserc. 4-4
697	717,3	716	02CC
770	649,3	648	0288
852	586,8	585	0249
941	531,3	530	0212
1209	413,5	412	019C
1336	374,2	372	0174
1477	338,5	337	0151

Figura 4-44: Calcolo delle Costanti del Timer

Allo stesso modo i semiperiodi delle altre frequenze compaiono in Figura 4-44. I corrispondenti valori esadecimali sono stati usati nel programma di Figura 4-41.

Esaminiamo ora alcuni miglioramenti a questo programma di base.

**ESERCIZIO 4-4:** Sono possibili alcuni miglioramenti quali, ad esempio, la precisione con cui si generano le frequenze. Con riferimento al capitolo 2 di questo libro o anche a un manuale hardware, si potrà notare che il timer 1 nel modo free running non genera un tono dell'esatta frequenza desiderata. Infatti esso aggiunge 1,5 microsecondi, oppure 2 microsecondi al valore del semiperiodo che è stato caricato nel registro del contatore. Ricalcolare le mezze frequenze che devono essere usate, supponendo che entrambi i timer, 1 e 2, aggiungano in media 1,75 microsecondi ad ogni semiperiodo.

**NOTA:** Non rifare i calcoli, la risposta è in Figura 4-44.

**ESERCIZIO 4-5:** Questo programma può essere funzionalmente migliorato anche aggiungendo un simbolo programmabile "silence". Ciò è usuale in alcuni paesi per comunicazioni internazionali, o all'interno di una compagnia per ottenere l'accesso ad una linea esterna. Bisogna innanzi tutto comporre alcuni digit per ottenere una linea, poi bisogna aspettare un determinato periodo prima di comporre il numero voluto. Inserire questo cambiamento nel programma di cui sopra.

Sotto è riportato un miglioramento hardware per eliminare delle frequenze.

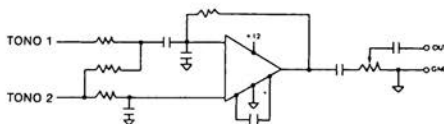


Figura 4-45: Miglioramento Hardware Suggesto per Annullamento di Frequenze

## PARTE 2:

### COMBINAZIONI DI TECNICHE

#### INTRODUZIONE

I programmi presentati in questo paragrafo, useranno una combinazione delle tecniche indicate in questo capitolo e saranno sviluppati per la scheda KIM. L'unica differenza significativa fra KIM e SYM al fine di questi esercizi, sarà la posizione del PIO nella mappa di memoria. Il lettore interessato è rimandato alla Figura 2-4 per la mappa di memoria reale del KIM. Poiché i programmi sono scritti in linguaggio assembler, facendo uso di label simboliche ed operandi, molti di essi saranno identici per il SYM. È solamente durante il processo di assembly (sia per mezzo di un assembler automatico come quello presentato nell'Appendice A, o per mezzo di un assembly manuale), cioè nel momento in cui si genera la rappresentazione esadecimale delle istruzioni, che compariranno differenze dovute a diverse assegnazioni di memoria.

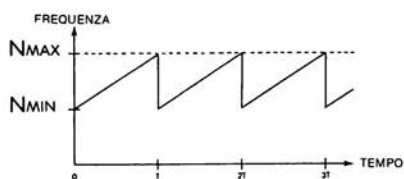


Figura 4-46: Suono di una Sirena



Figura 4-47: Diagramma di flusso per una Sirena-Rampa di Salita

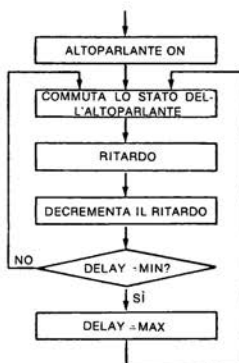


Figura 4-48: Stop ad  $N_{max}$

## GENERAZIONE DEL SUONO DI UNA SIRENA

La rappresentazione grafica del suono di una sirena è mostrata in Figura 4-46. Tale suono inizia alla frequenza minima  $N_{min}$ , e aumenta durante il tempo  $T$ , fino ad un valore massimo chiamato  $N_{max}$ . La frequenza del tono scende poi istantaneamente a  $N_{min}$  e risale progressivamente fino all'istante  $2T$  e così via. Il diagramma di flusso per la generazione di un suono di frequenza crescente è riportato in Figura 4-47. Inoltre la massima frequenza non potrà superare  $N_{max}$ , altrimenti il suono non potrà udirsi (oppure non potrà essere generato dall'altoparlante). Il diagramma di flusso per generare ripetitivamente la rampa è mostrato in Figura 4-48.

Il programma è riportato in Figura 4-49. Esso approssima la forma di Figura 4-46.

```

;SIREN
;
PA      = $1700
PAD     = $1701
;
0000: FF      DELAY    .BYP $FF
                        . = $40
0040: A9 01    LDA     #$01
0042: 8D 01 17 STA     PAD
0045: 8D 00 17 STA     PA
0048: EE 00 17 SWITCH  INC   PA
004B: A6 00    LDX     DELAY
004D: CA      LOOP    DEX
004E: D0 FD    BNE     LOOP
0050: C6 00    DEC     DELAY
0052: 4C 48 00 JMP     SWITCH
;

SYMBOL TABLE:
PA      1700      PAD      1701      DELAY      0000
SWITCH  0048      LOOP    004D
DONE

```

Figura 4-49: Programma per Sirena relativo al Flow-Chart di Figura 4-47

L'altoparlante è connesso al registro IORA (indirizzo di memoria 1700) nella posizione del bit 0. Esso può essere collegato direttamente. Per avere un suono migliore si consiglia il circuito di Figura 4-50. Il registro di direzione dati DDRA, di questo PIO, deve innanzi tutto, essere programmato in modo che il bit zero sia un'uscita.

```

LDA     #$01
STA     PAD      DDRA

```

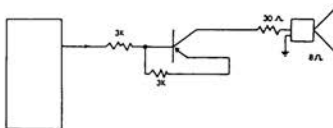


Figura 4-50: Collegamento di un Altoparlante (Migliorato)

Si può poi accendere l'altoparlante. L'accensione e lo spegnimento dell'altoparlante, si possono ottenere con un trucco di programmazione. Ciò consiste nell'istruzione INC. Tale istruzione incrementerà il contenuto di un apposito registro e genererà una successione di numeri che saranno alternativamente pari e dispari. Ciò assicura che il bit più a destra (il bit 0, a cui è collegato l'altoparlante) commuterà fra il valore 0 e 1. Questo permette l'accensione e lo spegnimento dell'altoparlante con una sola istruzione invece di due, che servirebbero se si dovesse caricare un pattern differente nell'accumulatore e trasferirlo poi ad ORA. Spegniamo ora l'altoparlante. Esso resterà spento per un periodo di tempo indicato dalla costante "DELAY". Il loop di ritardo è il seguente:

	STA	PA	VALORE INIZIALE IN ORA
SWITCH	INC	PA	
	LDX	DELAY	
LOOP	DEX		
	BNE	LOOP	

Una volta che DELAY è stato usato, viene decrementato:

DEC	DELAY
-----	-------

In questo modo, la prossima volta, il valore del ritardo sarà più piccolo e il tono avrà un passo più alto. L'altoparlante deve ora essere commutato:

JMP	SWITCH
-----	--------

Il programma precedente, implementa la rampa di salita del suono a sirena, come mostrato nel flow-chart di Figura 4-47.

**ESERCIZIO 4-7:** Completare il programma per il flow-chart di Figura 4-48, per generare rampe di salita successive e generare un vero suono a sirena.

**ESERCIZIO 4-8:** Scrivere un programma per sirena che salga in passo, poi scenda, poi salga di nuovo, etc.

## RILEVAMENTO DI UN IMPULSO IN INGRESSO

In questo programma si schiacerà un pulsante e si dovrà misurare la durata del periodo in cui il pulsante è mantenuto schiacciato, poi si suonerà n volte attraverso un altoparlante, dove n è il periodo, in secondi, durante il quale l'interruttore è stato schiacciato. L'altoparlante è collegato al bit 0 del IORA come nel programma precedente. L'interruttore, per semplicità di rilevamento, è collegato al bit 7 del IORA. Il collegamento è illustrato in Figura 4-51.

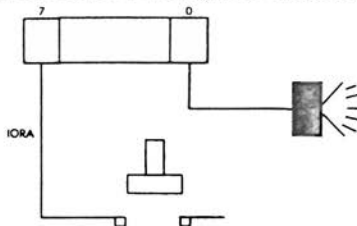


Figura 4-51: Collegamento di un Interruttore ed Altoparlante

Il diagramma di flusso del programma è mostrato in Figura 4-52. La durata del periodo durante il quale il pulsante è premuto, è misurata in unità di 0,25 secondi, poi è convertita in secondi. L'altoparlante è successivamente attivato e regolato.

**Note**

- COUNTER contiene "n" (numero di beep).
- N è la durata.

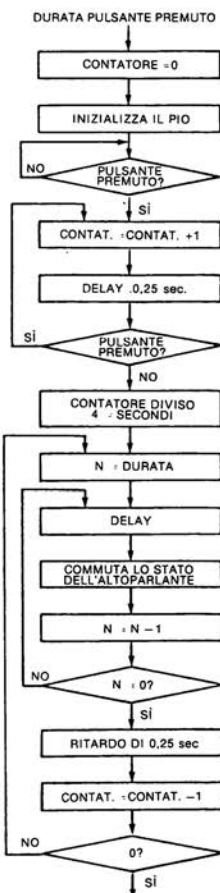


Figura 4.52: Diagramma di Flusso Dettagliato





## MISURA DELL'IMPULSO

In questo programma sarà misurato il tempo durante il quale il pulsante è stato premuto e si genererà un suono. Il numero di beep dovrà essere proporzionale alla durata della chiusura dell'interruttore.

Il diagramma di flusso di questo programma è analogo al precedente ed è riportato in Figura 4-54. Il programma corrispondente è mostrato in Figura 4-55.

Tale programma usa la subroutine DELAY che misura un ritardo di 0,25 secondi. Il diagramma di flusso di questa subroutine è mostrato in Figura 4-56. Il programma corrispondente è riportato in Figura 4-57.

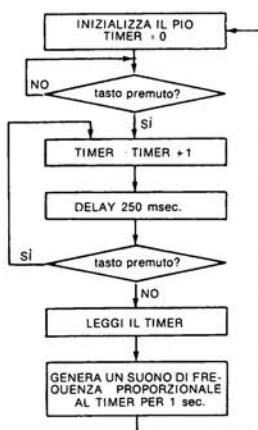


Figura 4-54: Misura del Tempo di Switch

```

FA      = $1700
FAD     = $1701
DL250   = $0090
FREQ    = $00C0
;
;
0000: 00      . = 00
;            . BYT $00
;            . = $40
  
```

Figura 4-55: Programma per la Misura del Tempo di Switch: Generazione del Tono

```

0040: A9 00          LDA #00
0042: B5 00          STA T
0044: B0 00 17       STA PA          :INIZIALIZZ. TEMPO
0047: A9 01          LDA #01
0049: B0 01 17       STA PAD          :BIT 0 FUORI
004C: AD 00 17       FOL LDA PA          :POLLING...
004F: 30 F8          BMI FOL          :NON SCHIACCIATO
0051: E6 00          CPT INC T          :INCREMENTA IL TEMPO
0053: 20 90 00       JSR DL250        :RITARDO DI 250 MS
0056: AD 00 17       LDA PA
0059: 10 F6          BFL CPT
005B: A5 00          HERE LDA T
005D: 0A             ASI A          :MPY PER 2
005E: 0A             ASI A          :MPY ANCORA PER 2
005F: 20 C0 00       JSR FREQ        :FA IL TONO
0062: 4C 5B 00       JMP HERE

SYMBOL TABLE:
PA          1700          PAD          1701          DL250          0090
FREQ        00C0          T            0000          FOL          004C
CPT          0051          HERE
DONE

:FA UN TONO. USA IL REGISTRO A
:ASSUME PA COME USCITA
:LA COSTANTE FREQUENZA NEL REGISTRO A IN INGRESSO
;
PA          =#1700
F           =#BF
;
;=#C0
00C0: B5 BF          FREQ STA F
00C2: A9 00          LDA #0
00C4: A2 B0          LDX #5B0        :COSTANTE DI DURATA
00C6: A4 BF          FL2 LDY F        :LA COSTANTE FREQUENZA IN Y
00C8: C8            FL1 INY
00C9: D0 FD          BNE FL1
00CB: 49 01          EOR #1
00CD: B0 00 17       STA PA          :TOGGLE PA0
00D0: E8            INX
00D1: D0 F3          BNE FL2
00D3: A5 BF          LDA F
00D5: 60            RTS

SYMBOL TABLE:
PA          1700          F            00BF          FREQ          00C0
FL2         00C6          FL1          00CB
DONE

```

Figura 4-55: Programma per il Tempo di Switch (continua)

**ESERCIZIO 4-9:** Il diagramma di flusso di Figura 4-55, è stato scritto in modo che ogni blocco corrisponda a una istruzione del programma di Figura 4-54. Usando questo flow-chart, o anche il programma, scrivere sulla sinistra di ogni blocco, la durata del ritardo da esso introdotto. Calcolare il ritardo interno totale di questa subroutine. Vale esattamente 250ms?

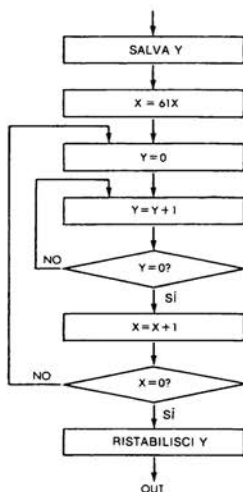


Figura 4-56: Diagramma di flusso per un Ritardo di 250ms

```

***** DL250 *****
;RITARDO DI 250 MILLISECONDI
;REG. Y NON INFLUENZATO
;

0090: 98          DL250   TAY          ;SALVA Y
0091: A2 3D          LDX          #3D
0093: A0 00          DL2   LDY          #0
0095: C8          DL1    INY
0096: D0 FD          BNE DL1          ;LOOP INTERNO
0098: E8          INX
0099: D0 F8          BNE DL2          ;LOOP PIÙ ESTERNO
009B: A8          TAY
009C: 60          RTS

SYMBOL TABLE:
DL250          0090          DL2          0093          DL1          0095

DONE
  
```

Figura 4-57: Ritardo di 250ms

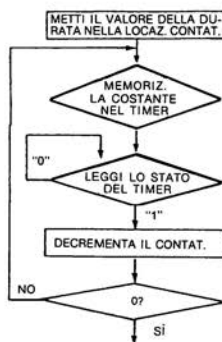


Figura 4-58: Diagramma di flusso TIME 10 (10Hz)

```

***** TIME10 *****
:RITARDO DI 1'10 DI SECONDO
:
TIMER      =%1707
D          =%9D
:
:          =%9E
009E: 86 9D   TIME10 STX D
00A0: A9 62   T0     LDA #%62 :98 DECIMALE
00A2: 8D 07 17 STA TIMER
00A5: AD 07 17 T1     LDA TIMER
00A8: 10 FB   BPL T1
00AA: C6 9D   DEC D
00AC: D0 F2   RNE T0
00AE: 60     RTS

SYMBOL TABLE:
TIMER      1707      D          009D      TIME10      009E
T0         00A0      T1         00A5
  
```

Figura 4-59: Generazione di un Ritardo di 0,1 Secondi

## UN SEMPLICE PROGRAMMA MUSICALE

Come gradino preliminare per suonare della musica, generiamo ora un suono con l'altoparlante impiegando un ritardo programmato. Il flow-chart è riportato in Figura 4-58 e la subroutine di ritardo, è mostrata in Figura 4-59. Prima di usare la subroutine bisogna caricare la costante F con l'opportuna durata del ritardo, che determinerà la frequenza del suono. Allo scopo di generare musica che abbia qualche rassomiglianza ai motivi attuali, è necessario generare

un suono di frequenza specifica e controllarne anche la durata. I simboli musicali usati per indicare la durata di un tono, sono riportati di seguito:

Simboli Musicali

(. = +50%)

$\text{♩} = 1$ 
  
 $\text{♪} = 2$ 
  
 $\text{♫} = 3$ 
  
 $\text{♬} = 4$ 
  
 $\text{♭} = 6$ 
  
 $\text{♮} = 8$ 
  
 $\text{♯} = 12$

Il punto che può seguire una nota indica la durata più 50%. In tutto ci sono sette possibili durate. Inoltre è necessario rappresentare una "pausa". Tale informazione richiederà come minimo tre bit in un formato *codificato*, o anche quattro bit in un formato *non codificato*. (Un formato non codificato, è quello in cui i valori 1, 2, 3, 4, 6, 8 e 12 sono rappresentati nella loro forma binaria). Per rappresentare le note di un'ottava, bisogna rappresentare A, B, C, D, E, F, G, come pure le sei mezze note fra di esse. Ciò significa un totale di 13 chiavi per ogni ottava. Se si vuole usare più di un'ottava, bisogna poi riservare un intero byte per rappresentare un tono. Se il lettore, ha una quantità di memoria disponibile limitata sulla sua piastra, può desiderare di limitare i propri motivi a 16 possibili chiavi e potrà quindi usare una codifica dove la metà sinistra di ogni byte rappresenta la durata e la parte destra rappresenta le note.

Qui suoneremo semplici motivi e useremo una semplice tecnica di codifica, dove un intero byte è riservato alla durata ed un altro alla frequenza della nota. Tre esempi, una Sonatina di Mozart, una Corale di Bach e una canzone popolare dei bambini, sono riportati in Figura 4-60, 4-61 e 4-62.

Indirizzo	Durata	F	Note
00	09	20	la  A
2	04	4F	do#  C#
4	04	6B	mi  E
6	05	12	sol#  G#
8	01	20	la  A
A	01	39	si  B
C	0F	20	la  A
E	02	00	
12	09	7C	fa#  F#
12	04	6B	mi  E
14	04	91	la  A
16	04	6B	mi  E
18	04	59	re'  D
1A	09	4F	do#  C#
1C	00	00	
1E			
20			

Figura 4-60: Sonatina di Mozart

Il diagramma di flusso per il corrispondente programma musicale è riportato in Figura 4-63 ed il programma appare in Figura 4-64.

Un timer a 0,1 secondo è una semplice subroutine preliminare che genererà un ritardo di 0,1 secondi (vedi Figure 4-58, 4-59).

Indirizzo	Durata	F	Note
00	88	44	<i>do</i> C
02	06	59	<i>ré</i> D
04	06	6B	<i>mi</i> E
06	88	83	<i>sol</i> G
08	06	74	<i>fa</i> F
A	06	74	<i>fa</i> F
C	88	91	<i>la</i> A
E	06	83	<i>sol</i> G
10	06	83	<i>sol</i> G
12	88	A3	<i>do</i> C
14	06	9E	<i>si</i> B
16	06	A3	<i>do</i> C
18	06	83	<i>sol</i> G
1A	06	6B	<i>mi</i> E
1C	06	44	<i>do</i> C
1E	88	59	<i>ré</i> D
20	06	6B	<i>mi</i> E
22	06	20	<i>la</i> A
24	88	83	<i>sol</i> G
26	06	74	<i>fa</i> F
28	06	6B	<i>mi</i> E
2A	88	59	<i>ré</i> D
2C	06	44	<i>do</i> C
2E	06	04	<i>sol</i> G
30	06	44	<i>do</i> C
32	88	39	<i>si</i> B
34	06	44	<i>do</i> C
36	06	6B	<i>mi</i> E
38	06	83	<i>sol</i> G
3A	0E	A3	<i>do</i> C
3C	0E	44	<i>do</i> C
3E	00	00	

Figura 4-61: Corale di Bach























Indirizzo	Durata	F	Note
0	04	44	do  C
2	04	44	do  C
4	04	44	do  C
6	04	59	ré  D
8	09	6B	mi  E
A	09	59	ré  D
C	04	44	do  C
E	04	6B	mi  E
10	04	59	ré  D
12	04	59	ré  D
14	09	44	do  C
16	10	00	
18	04	44	do  C
1A	04	44	do  C
1C	04	44	do  C
1E	04	59	ré  D
20	09	6B	mi  E
22	09	59	ré  D
24	04	44	do  C
26	04	6B	mi  E
28	04	59	ré  D
2A	04	59	ré  D
2C	09	44	do  C
2E	00	00	

Figura 4-62: "Al Chiaro di Luna"

ESERCIZIO 4-10: Verificare se la subroutine implementa esattamente un ritardo di 0,1 secondi. Verificare la durata di ogni istruzione e il numero di volte che il loop viene eseguito. Calcolare il corrispondente ritardo.

## CONTROLLO DEL TRAFFICO CON IL KIM

In Figura 4-66 è riportato un possibile collegamento per la simulazione di un controllo del traffico. Esso è munito di interruttori in ogni direzione, i quali saranno usati per indicare la presenza di una macchina o anche una chiamata pedonale

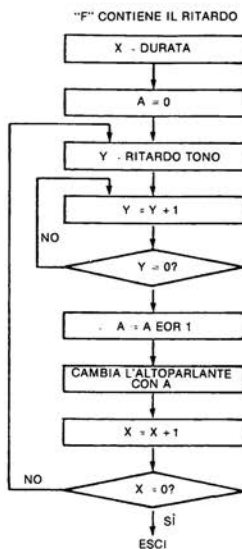


Figura 4-63: Diagramma di flusso di Generazione del Suono

```

***** PLAY A TUNE *****
FA      = $1700
PAD     = $1701
TIMER   = $1707
        = 00
ADDRS   = .+2
TEMP    = .+1
YSAVE   = .+1
F        = .+1
        = $10
0010: A9 31  TIME20 LDA #$31
0012: 8D 07 17 STA TIMER
0015: 2C 07 17 T1 BIT TIMER
0018: 10 FB BPL T1
001A: CA DEX
001B: D0 F3 BNE TIME20
001D: 60 RTS

```

Figura 4-64: Suono di un Accordo



```

;
;=$20
0020: B4 03   FREQT   STY YSAVE
0022: B5 04           STA F
0024: A9 31   FT0     LDA #$31
0026: BD 07 17   STA TIMER :PARTENZA TIMER (1'20 SEC.)
0029: A4 04   FT1     LDY F
002B: C8       FT2     INY
002C: D0 FD           BNE FT2
002E: EE 00 17   INC FA :COMMUTA L'ALTOPARLANTE
0031: 2C 07 17   BIT TIMER :È PASSATO IL TEMPO?
0034: 10 F3           BPL FT1 :NO: VA AVANTI
0036: CA       FT3     DEX
0037: D0 EB           BNE FT0
0039: A4 03   LDY YSAVE
003B: 60       RTS
;
;=$40
0040: A2 0F   START   LDX #$0F
0042: 7A       TXS
0043: A9 00       LDA #$00
0045: BD FA 17   STA $17FA
0048: BD FE 17   STA $17FE
004B: A9 1C       LDA #$1C
004D: BD FB 17   STA $17FB
0050: BD FF 17   STA $17FF :VETTORE DI INTERRUPT
0053: A9 01       LDA #$01
0055: B1 01 17   STA PAD :PAO È UN'USCITA
005B: A0 00   DACAF0  LDY #$00
005A: B1 00   NEXT    LDA (ADDRS),Y
005C: B5 02       STA TEMP
005E: 29 7F       AND $7F :DURATA
0060: A6       TAX
0061: F0 F5   DACAF0  BEQ DACAF0
0063: C8       INY
0064: B1 00   LDA (ADDRS),Y
0065: F0 10   BEQ TIME
006B: 20 29 00   JCR FREUT
006B: 24 02       BIT TEMP
006D: 30 05       BMI AFTER
006F: A2 02       LDX #$02
0071: 20 10 00   JSR TIME20
0074: C8       AFTER    INY
0075: AC 7A 00   IMM: $7A :0
007B: 20 10 00   TIME   JSR TIME20
007B: F0 F7       BEQ AFTER

```

SYMBOL TABLE:

FA	1700	PAD	1701	TIMER	1707
ADDRS	0000	TEMP	0002	YSAVE	0003
F	0004	TIME20	0010	T1	0015
FREQT	0020	FT0	0024	FT1	0029
FT2	002B	FT3	0036	START	0040
DACAF0	005B	NEXT	005A	AFTER	0074
TDNE	007B				

Figura 4-64: Suono di un Accordo (continua)

ESERCIZIO 4-11: Scrivere un programma di controllo del traffico che rispetti le seguenti specifiche:

- Durata minima del giallo: 3 secondi
- Quando è rilevata la presenza di un'automobile (mantenendo premuto uno degli interruttori) estendere la durata del verde di tre secondi.
- Massima durata del verde in ogni direzione: 2 minuti, se c'è una richiesta nella direzione opposta.
- Lampeggio del semaforo durante la notte (un'indicazione della notte è data da un interruttore separato).
- Un possibile diagramma di flusso per questo programma è riportato in Figura 4-65. Scrivere il corrispondente programma.

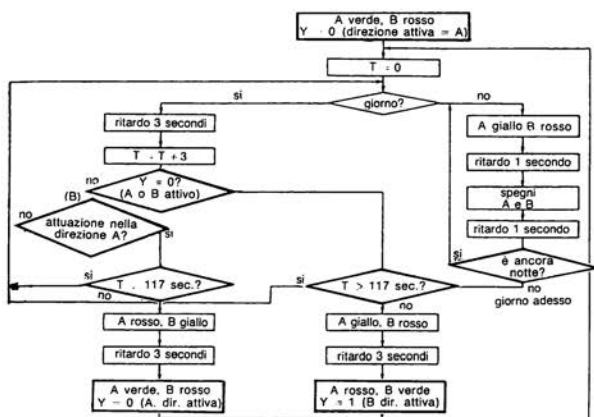


Figura 4-65: Diagramma di flusso del Traffico

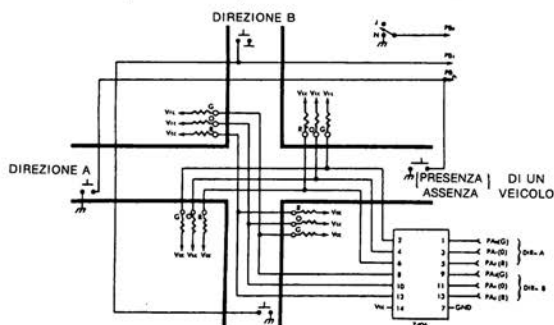


Figura 4-66: Controllore del Traffico

## STUDIO DELLA TABELLA DI MOLTIPLICAZIONE

Come esercizio finale, tale programma insegnerà la tabella della moltiplicazione. Il programma farà lampeggiare un LED o l'altoparlante  $n$  volte, con  $n$  compreso fra 1 e 10, poi aspetterà 2 secondi, e lampeggerà ancora  $p$  volte, con  $p$  compreso fra 1 e 10.

L'utilizzatore deve poi pigiare  $n$  volte un interruttore a pulsante per inserire la sua risposta. Un consenso sonoro può essere fornito dall'altoparlante. L'utilizzatore termina la sua risposta non pigiando l'interruttore per 3 o più secondi. Se la risposta era corretta, il LED resterà acceso per cinque secondi. Se la risposta non era corretta il LED lampeggerà.

*ESERCIZIO 4-12: Progettare il flow-chart corrispondente e scrivere il programma (Questo programma è semplice ma un po' più lungo di molti dei precedenti. Se si vuole veramente la risposta, questa è riportata in Appendice B).*

## SOMMARIO

Si sono ora collegati semplici dispositivi d'ingresso-uscita ad un sistema 6502. Abbiamo imparato a realizzare semplici interfacce hardware e come sviluppare semplici applicazioni software per rilevare e controllare un ambiente esterno. Sebbene la complessità delle applicazioni qui presentate è stata limitata, si possono sviluppare applicazioni più complesse usando lo stesso semplice hardware. Siamo ora pronti a continuare con programmi e interfacce più complesse nel capitolo 5: Applicazioni Industriali ed Home.



## CAPITOLO 5

# APPLICAZIONI INDUSTRIALI E DOMESTICHE

### INTRODUZIONE

Nel Capitolo 4 sono stati presentati gli artifici di base per la connessione di semplici dispositivi alla scheda microcomputer, basata sul 6502, e per lo sviluppo delle applicazioni software fondamentali. Nel corso di questo capitolo verranno interfacciati alla scheda del 6502 dei dispositivi più complessi e verranno sviluppate delle applicazioni software di complessità crescente. Le applicazioni presentate sono tipicamente delle situazioni di controllo industriale ed home. Nel capitolo successivo, le periferiche del microcomputer saranno interfacciate alla scheda del 6502.

Una simulazione del controllo del traffico sarà la prima applicazione presentata. I semafori saranno simulati da LED, assemblati sulla scheda e verranno sviluppati dei programmi di applicazione di complessità crescente. La presenza di automobili sarà rivelata da rivelatori ad anello, normalmente incassati nell'asfalto, qui simulati da interruttori a pulsante. Le conoscenze richieste per lo sviluppo di queste interfacce hardware e software sono quelle richieste da un effettivo sistema di controllo Industriale.

Successivamente verrà interfacciata al sistema una matrice LED di 5 x 7 punti. Si tratta di una tecnica spesso impiegata nel display di dati. Le matrici di punti vengono utilizzate normalmente, non per i LED, ma per rappresentare caratteri su uno schermo televisivo, oppure su una stampante a matrice di punti. Questa matrice di punti verrà utilizzata per mostrare le posizioni effettive degli interruttori, così come sono rivelati dalla scheda 6502.

Inoltre verranno generati dei toni con un altoparlante per sviluppare semplici programmi per generare della musica. L'insieme di interruttori verrà impiegato per indicare la nota che si vuole eseguire. L'esperienza acquisita nel controllo del suono dell'altoparlante verrà utilizzata dal programma successivo per generare il suono di una sirena.

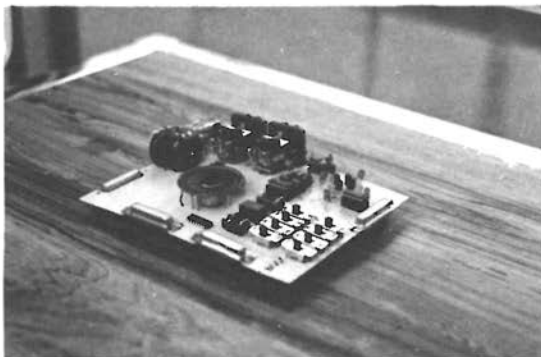
Il successivo programma di applicazione realizza un sistema di allarme contro scassinatori per una casa oppure un fabbricato. Un raggio luminoso verrà impiegato come dispositivo di rivelazione di un eventuale intruso. Ogni interruzione del raggio di luce farà scattare l'allarme, mettendo in funzione l'altoparlante. Per esercizio saranno proposti numerosi miglioramenti addizionali.

Quindi si controllerà con il computer la velocità di un comune motore in c.c. e si noterà che è molto semplice controllare la velocità di un motore con tecniche digitali. Verranno presentate queste tecniche e l'interfaccia hardware richiesta.

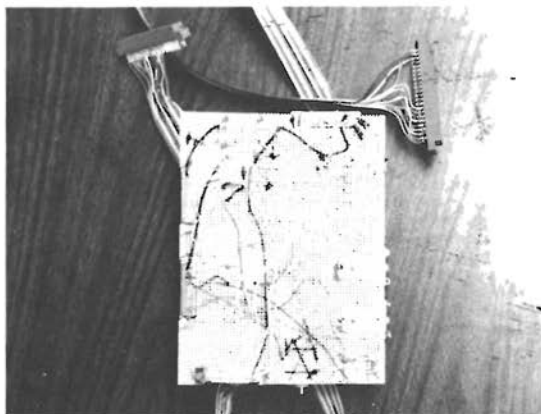
Nell'applicazione successiva si conatterà alla scheda microprocessore un dispositivo sensore di calore e la misura di temperatura verrà rivelata come suono udibile. A temperatura più alta corrisponderà una maggiore altezza del tono. Questo introdurrà il concetto di conversione analogico-digitale e quindi verranno presentate le tecniche effettive, hardware e software, coinvolte in questa conversione.

Il lettore verrà incoraggiato alla realizzazione effettiva della scheda di applicazioni #2 richiesta dal programma di questo capitolo. Tutti i componenti impiegati in questa scheda sono di basso costo e normalmente disponibili in un negozio di componenti elettronici (probabilmente

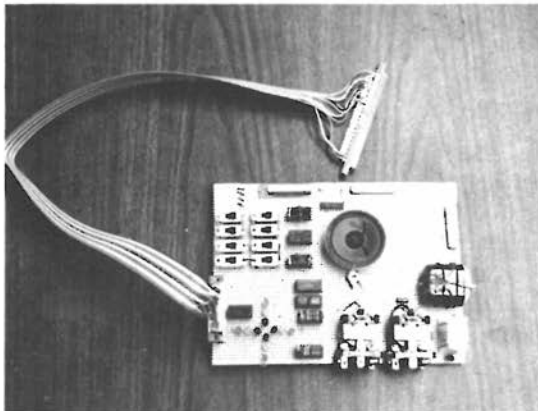
eccetto il convertitore digitale-analogico che può essere ordinato da un distributore). Le Figure 5-1, 5-2 e 5-3 mostrano le fotografie delle schede reali.



*Figura 5-1: La Scheda di Applicazione #2*



*Figura 5-2: Il retro mostra i Collegamenti wire-wrap*



*Figura 5-3: Connessione alla scheda dei Cavi di Applicazione*

In vista del limitato numero di porte, normalmente disponibili sull'uscita della scheda micro-processore, verranno installati quattro connettori, contrassegnati H1, H2, H3 ed H4 che saranno posizionati sulla scheda stessa per facilitare le connessioni e per evitare ricuciture tra programmi. Questi connettori sono stati progettati per accoppiarsi direttamente ai cavi di connessione esterna SYM ma potrebbero essere facilmente adattati all'uscita di altre schede micro-computer. Per ogni applicazione sarà necessario inserire uno o due dei cavi di uscita provenienti dalla scheda ai connettori corrispondenti mostrati sulla scheda delle applicazioni. I dettagli di ogni applicazione saranno indicati all'inizio dell'applicazione stessa.

La disposizione dei componenti per la scheda è mostrata in Figura 5-4. Invece i dettagli dei connettori sono riportati nelle Figure 5-5A e B. I dettagli di ogni connessione sono riportati in corrispondenza del paragrafo relativo.

Per la connessione dei terminali ai pin sul retro della scheda è stata impiegata una tecnica wire-wrap, come mostrato in Figura 5-2. Naturalmente è possibile saldare i fili. Non si trascurino le comuni precauzioni per la manipolazione dei circuiti LSI: tutti gli strumenti (compresi voi stessi) devono essere collegati a terra. Infine il potenziometro trimmer (resistore variabile), connesso in serie all'altoparlante, non dovrebbe essere posto a zero. In caso affermativo il potenziometro potrebbe bruciare all'atto di applicazione dell'alimentazione all'altoparlante, nel caso di una scheda simile alla SYM, dove l'altoparlante verrà connesso ad una delle uscite buffered (inoltre si può bruciare il transistor d'uscita). Per questo motivo si raccomanda di inserire un resistore addizionale in serie all'altoparlante.

Lo scopo di questo capitolo è di insegnare le applicazioni tecniche effettive che consentano di sviluppare applicazioni home di notevole complessità o di risolvere problemi di controllo industriale in un ambiente reale. Alla fine di questo capitolo si saranno acquisiti tutti gli artifici di

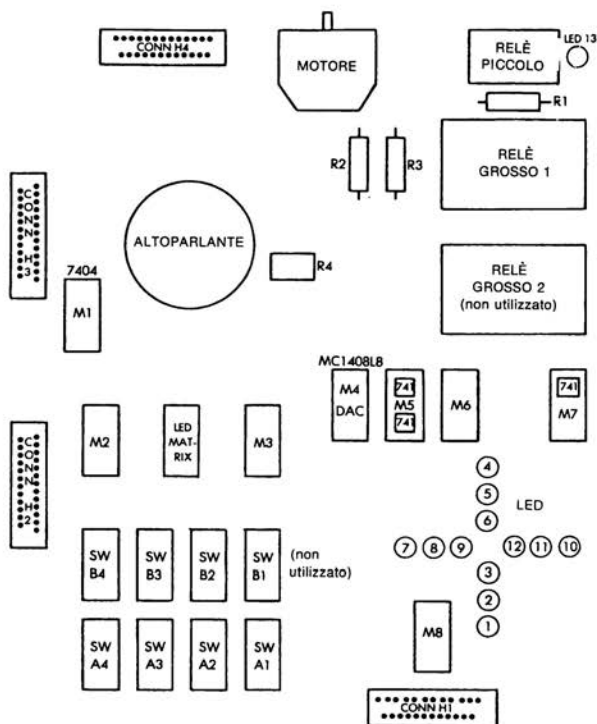


Figura 5-4: Layout della scheda

base richiesti per uno sviluppo indipendente di applicazioni complesse. Se si dovessero incontrare problemi specifici di interfacciamento si faccia riferimento a "Tecniche di interfacciamento dei microprocessori" Codice 3148.

**NOTA IMPORTANTE:** Per impiegare più di una linea di ingresso-uscita, si opera il by-pass del transistor 1 (più al centro) delle quattro porte buffered del SYM.

I programmi presentati in questo capitolo sono stati progettati per essere migliorati. Il lettore attento noterà che sono possibili molti miglioramenti per quanto concerne lo stile. Tali miglioramenti sono proposti e descritti nella parte di esercizi riportata alla fine di ogni applicazione. Nella lettura dei programmi si suggerisce di osservare i miglioramenti possibili. Comunque so-



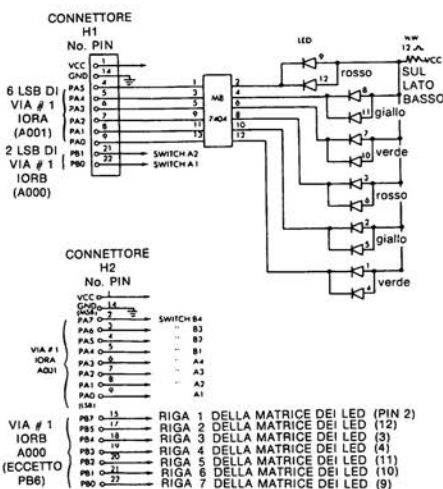


Figura 5-5A: Connettori H1 ed H2

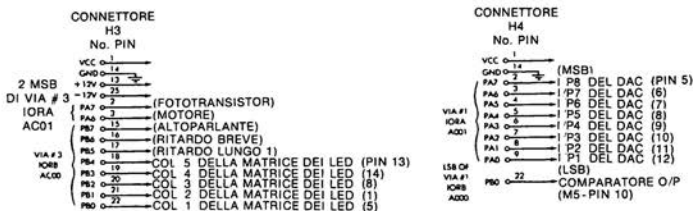


Figura 5-5B: Connettori H3 ed H4

Io al capitolo successivo verranno presentati programmi ottimizzati, una volta che saranno risolti tutti i problemi.

Inoltre, nel corso del capitolo, verrà proposto un grande numero di esercizi. Si raccomanda vivamente di risolvere la maggior parte di questi esercizi sia su carta che su una scheda microcomputer. Essi sono stati accuratamente preparati per assicurare che i concetti presentati nei capitoli precedenti siano effettivamente assimilati e che essi possano essere utilizzati in modo creativo. Se si è in grado di risolvere gli esercizi senza consultare il libro significa che si è effettivamente imparato a risolvere i problemi applicativi.

## UN SISTEMA DI CONTROLLO DEL TRAFFICO

Si intendono sviluppare dei programmi per il controllo di un incrocio simulato. La Figura 5-6 riporta lo schema dell'incrocio. Esistono due direzioni di flusso del traffico, indicate con A e B, che nel gergo di controllo del traffico vengono chiamate "fasi". I due semafori di entrambe le direzioni di una fase, per esempio dell'arteria A, mostreranno contemporaneamente lo stesso colore (verde, giallo o rosso). Analogamente per gli altri due semafori della fase B. Questi quattro semafori saranno simulati su una scheda da quattro insiemi di LED verdi, gialli e rossi. Inoltre si assumerà che i rivelatori di veicolo ad anello siano incassati nell'asfalto nelle quattro posizioni contrassegnate A-1, A-2, B-1 e B-2 nello schema di Figura 5-6. Essi verranno chiamati "rivelatori ad anello" e la loro funzione verrà spiegata in seguito.

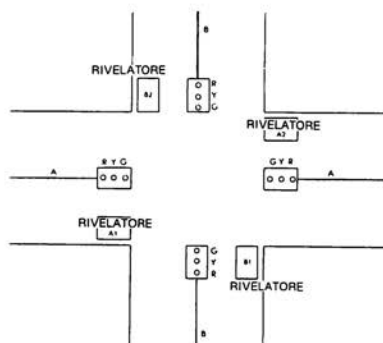


Figura 5-6: Il Sistema di Controllo del Traffico

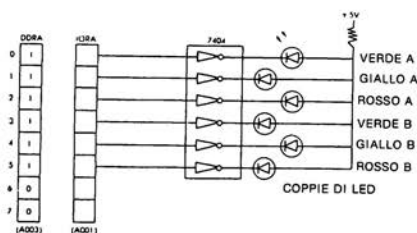


Figura 5-7: Connessione dei LED

Preliminarmente si esamini la connessione hardware del "semaforo" simulatore (di LED) al sistema microprocessore. Con riferimento alla Figura 5-7, è stato connesso un driver 7404 al registro IORA del 6522 =1. Le coppie di LED sono riportate sulla destra dell'illustrazione. Per motivi di chiarezza viene riportato solo un LED per ogni linea. Infatti su ogni linea in realtà sono due semafori per ogni fase. La connessione effettiva è riportata in Figura 5-8. Per assegnare i 6 bit di basso ordine di IORA all'uscita, il registro di direzione DDRA, che appare alla sua sinistra, sarà caricato con la struttura di bit "00111111". È necessario un driver (il 7404) per fornire la corrente sufficiente ad illuminare i LED.

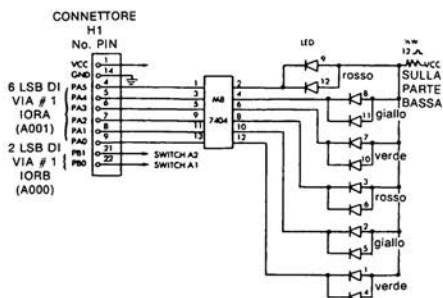


Figura 5-8: Connessione reale dei LED

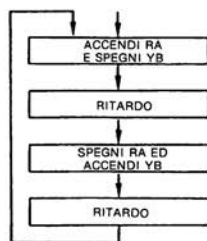


Figura 5-9: Funzionamento Notturno

Si svilupperanno ora i programmi per diversi algoritmi di controllo del traffico. Si possono distinguere due casi fondamentali: il funzionamento notturno (semafori lampeggianti) ed il funzionamento durante il giorno.

(Connessione: connettore A al connettore H1-)									
0100	A9	3F		NIGHT	LDA	#\$3F	*		
0102	8D	03	A0		STA	\$A003		Poni VIA #1 DDRA = 3F per il modo di uscita	
0105	A5	02		NIT2	LDA	#\$02			
0107	8D	01	A0		STA	\$A001		Accendi il giallo in una direzione	
A10A	A9	FC			LDA	\$FC		- conteggio.	
010C	85	00			STA	\$00		Poni conteggio di DLYA = \$FC (cioè -4) nella	
								locazione \$0000	
010E	20	20	01		JSR	DLYA		Chiama DLYA	
0111	A9	20			LDA	#\$20			
0113	8D	01	A0		STA	\$A001		Accendi la luce verde nell'altra direzione	
0116	A9	FC			LDA	#\$FC			
0118	85	00			STA	\$00		Poni conteggio di DLYA = \$FC alla locazione	
								\$0000	
011A	20	20	01		JSR	DLYA		Chiama DLYA	
011D	4C	05	01		JMP	NIT2		Ripeti	

Subroutine DLYA: Questa subroutine preleva l'indice dalla locazione 0000, opera finché questo indice non è stato incrementato da un valore negativo pre-impostato al valore 0. L'indice pre-impostato viene impiegato per controllare la lunghezza del ritardo.

0120	A2	9D	DLYA	LDX	#\$9D		
0122	A0	71	LPXA	LDY	#\$71		
0124	C8		LPYA	INY			
0125	C0	00		CPY	#\$00		
0127	30	FB		BMI	LPYA		
0129	E8			INX			
012A	E0	00		CPX	#\$00		
012C	30	F4		BMI	LPXA		
012E	E6	00		INC	\$00		
							Incrementa il conteggio del ritardo ogni volta
							che è completato il ciclo di ritardo esterno
0130	A5	00		LDA	\$00		
0132	C9	00		CMP	#\$00		
0134	30	EA		BMI	DLYA		
0136	80			RST			Ricicla finché l'indice è uguale a 0

Figura 5-10: Simulazione del semaforo: Modo Notturno (Programma 5-1)

## Funzionamento notturno

Si tratta del funzionamento più semplice: i semafori lampeggiano rosso in una direzione e giallo nell'altra. Questa strategia di controllo viene impiegata per gli incroci isolati a basso livello di traffico notturno. La Figura 5-9 riporta il diagramma di flusso corrispondente a questo algoritmo. Si assume che il rosso di una direzione ed il giallo dell'altra siano accesi e spenti contemporaneamente. Essi sono mantenuti accesi o spenti per un tempo fisso chiamato "DELAY". La Figura 5-10 riporta il programma corrispondente a questo diagramma di flusso. Si tratta di un programma principale chiamato "NIGHT" e di una subroutine di ritardo chiamata "DLYA". Si osservi il programma.

Con riferimento alla Figura 5-7, il registro di Direzione Dati del 6522 #1 deve prima essere configurato correttamente in modo che i sei bit di basso ordine di IORA siano le uscite che pilotano i LED. Questo DDRA si trova nella locazione di memoria A003 ed IORA nella locazione A001 (con riferimento alla Figura 3-6 della mappa di memoria del 6522).

Le prime due istruzioni caricano i contenuti richiesti nel Registro di Direzione Dati:

```
NIGHT    LDA    #$3F
          STA    $A003    PONI DDRA
```

Quindi occorre depositare semplicemente la struttura appropriata nel registro IORA per ac-

cendere o spegnere i LED richiesti. La struttura richiesta per l'indirizzamento di ogni coppia di LED è riportata in Figura 5-11.

BINARIO	ESADEC.	SEMAFORO
00000001	01	VERDE A
00000010	02	GIALLO A
00000100	04	ROSSO A
00001000	08	VERDE B
00010000	10	GIALLO B
00100000	20	ROSSO B

Figura 5-11: Schema per indirizzare le coppie di LED

Le due righe successive del programma accendono il giallo per A depositando il valore esadecimale "02" nel registro IORA.

```
NIT2    LDA    #02
        STA    $A001    PONI IORA
```

Si deve quindi realizzare un ritardo. Il valore del ritardo viene caricato nell'accumulatore e quindi memorizzato alla locazione di memoria «00» dove sarà prelevato dalla routine di ritardo. Si verifica quindi un salto della subroutine a DLYA.

```
LDA    #$FC
STA    $00
JSR    DLYA
```

Una volta trascorso il ritardo specificato, il valore esadecimale "20" viene depositato in IORA. Questo farà spegnere il giallo nella direzione A ed accendere contemporaneamente il rosso nella direzione B. Come prima la durata del ritardo viene caricata nella locazione di memoria "0" e si verifica un nuovo salto alla subroutine DLYA:

```
LDA    #$20
STA    $A001
LDA    #$FC
STA    $00
JSR    DLYA
```

Infine, dopo un ritardo ben preciso, il programma ritorna alla locazione NIT2 dove accende YA e spegne RB:

```
JMP    NIT2
```

A questo punto il funzionamento del programma è completamente definito. Si esamini la subroutine di ritardo: il ciclo di ritardo inizia col caricamento di un registro o di una locazione di memoria con un valore opportuno e quindi si esegue l'incremento o decremento fino a raggiungere un valore fisso. Poiché si presume che il lettore abbia familiarità con la tecnica di decremento si utilizzerà di seguito la tecnica alternativa dell'incremento. Comunque essa richiede qualche istruzione in più. In un esercizio, riportato alla fine del paragrafo, si suggerirà un miglioramento. La Figura 5-12 riporta la routine di ritardo. Poiché il ritardo da realizzare è dell'ordine delle decine di secondi, non può essere realizzato con un ciclo singolo. Un ciclo di ritardo singolo carica un registro con il valore 255 (esadecimale FF) e lo incrementa o decrementa. Il ritardo risultante non è sufficiente. Per la realizzazione di ritardi più lunghi si utilizzeranno dei cicli annidati: un ciclo di ritardo interno ed almeno un ciclo esterno, che verrà eseguito ogni

volta che si esce dal ciclo interno. Si osservi il programma: il registro X viene utilizzato come contatore del ciclo esterno. Esso viene caricato con il valore esadecimale 9D. Questo valore verrà giustificato in seguito:

```
DLYA      LDX      #$9D
```

La seconda istruzione del programma carica il registro X con il valore esadecimale 71. Y è il contatore del ciclo interno:

```
LPXA      LDY      #$71
```

Le tre istruzioni successive realizzano il ciclo di ritardo interno:

```
LPYA      INY
          CPY      #$00
          BMI      LPYA
```

Y viene incrementato finché non raggiunge il valore 0. Ogni volta che si esce dal ciclo di ritardo interno (cioè che Y raggiunge il valore 0) il contatore esterno X viene incrementato. Questa è la sesta istruzione del programma:

```
      INX
```

Ogni volta che X viene incrementato, viene confrontato con il valore 0 e, non appena si raggiunge questo valore, si ritorna indietro a LPXA cioè all'inizio del ciclo di ritardo interno:

```
      CPX      #$00
      BMI      LPXA
```

Alla fine il ritardo risultante sarà dato dal numero di esecuzioni del ciclo interno per quello del ciclo esterno.

Ogni volta che si esce dal ciclo esterno, il contatore di ritardo globale, alla locazione 00 viene incrementato di una unità:

```
      INC      $00
```

Questo è un terzo ciclo di ritardo. I contenuti della locazione di memoria 00 vengono confrontati con 00 ogni volta che vengono incrementati. Quando si raggiunge il valore 00 si esce dalla routine. In questo caso si ritorna indietro alla locazione DLYA, cioè all'inizio del ciclo precedente di ritardo per eseguire ancora la procedura:

```
      LDA      $00
      CMP      #$00
      BMI      DLYA
      RTS
```

La Figura 5-12 mostra la struttura globale del programma, con i suoi tre cicli di ritardo annidati e la sequenza delle istruzioni. Il ritardo globale è uguale al prodotto dei contenuti della locazione di memoria 00 per il numero di esecuzioni del ciclo interno per quelle del ciclo esterno. Si vuole calcolare la durata del ciclo totale con riferimento alla sequenza delle istruzioni riportata in Figura 5-12. Si consideri il ciclo interno: ogni volta vengono eseguite tre istruzioni impieganti sette microsecondi. Per semplicità si supponga di richiedere al ciclo interno di generare un ritardo di circa 1 millisecondo. Il ciclo esterno #1 genererà un ritardo di 100000 microsecondi (0,1 sec).

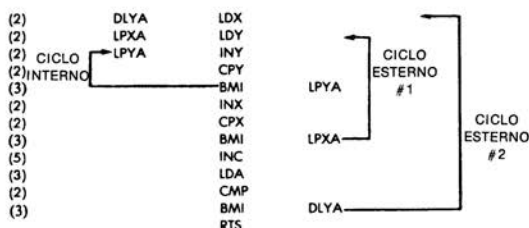


Figura 5-12: Disposizione dei Cicli

Si inizi con il valore "80" nel registro Y (valore esadecimale). Questo numero, in binario, equivale a 128, la metà del campo che può essere ottenuto con 8 bit. Eseguendo il ciclo interno si incrementerà Y 128 volte. La durata totale del ciclo sarà perciò  $7 \times 128 = 896$  microsecondi. Poiché si vuole ottenere un ritardo di circa 1.024 microsecondi, è necessario modificare il valore caricato nel registro Y. Si vuole calcolare il valore N tale che  $N \times 7 = 1000$ . N deve perciò essere uguale a  $1000 \div 7 = 142,86$ . L'intero più vicino è 143. Poiché, in questa particolare subroutine di ritardo, stiamo incrementando il valore contenuto in Y, invece di decrementarlo, vogliamo caricare in Y il valore  $256 - 143 = 113$  in decimale, ovvero 71 in esadecimale.

Calcoliamo ora la durata del ritardo introdotto dal ciclo di ritardo più esterno #1. Un attraversamento del ciclo di ritardo più esterno genererà un ritardo uguale alla durata della prima istruzione del programma (di indirizzo DLYA) più la durata del ciclo di ritardo più interno, più le tre istruzioni successive fino a comprendere la diramazione BMI LPXA. La durata vale:

$$2 + 7 \times 143 + 7 = 1010 \text{ microsecondi}$$

Noi vogliamo realizzare, con questo ciclo di ritardo più esterno, un ritardo di 0,1 secondi, cioè 100000 microsecondi. Quindi il numero di volte P che si deve eseguire questo ciclo sarà tale che  $1010 \times P = 100000$ . P deve perciò essere uguale a  $100000 \div 1010 = 99$ .

Inoltre, poiché stiamo realizzando una tecnica di incremento, il numero posto in X deve essere tale che esso venga incrementato esattamente 99 volte prima di diventare "00". Il numero da caricare in X deve perciò essere uguale a  $256 - 99 = 157$  in decimale, ovvero 9D in esadecimale. Verifichiamo ora la durata del ritardo realizzato complessivamente. Il ritardo del ciclo più esterno è uguale a  $99 \times 1010 = 99990$  microsecondi. Le quattro istruzioni che rimangono da eseguire alla fine della subroutine DLYA rappresentano una durata di  $5 + 3 + 2 + 3 = 13$  microsecondi. Per la prima istruzione di DLYA devono esser sommati 2  $\mu s$ .

Il ritardo totale prodotto da un attraversamento completo di DLYA è perciò  $99990 + 15 = 100005$  microsecondi. Questo rappresenta un valore molto vicino al ritardo di 0,1 secondi.

**NOTA:** Si ricordi che questa subroutine utilizza una tecnica di incremento. Il numero da caricare nella locazione di memoria 00 determinerà il numero di decimi di secondo del ritardo che la subroutine introdurrà. Comunque il numero da caricare nella locazione 00 sarà il complemento del numero effettivo di decimi di secondo poiché esso viene incrementato finché non raggiunge 0. In altre parole, per ottenere un ritardo di 0,4 secondi, non si deve caricare il valore 4 nella locazione 00 ma il valore  $256 - 4 = 252$  in decimale, o FC esadecimale. Questo è ciò che è stato eseguito nel programma di Figura 5-10 (algoritmo del funzionamento notturno).

Si vuole ora migliorare questa routine di ritardo.

**ESERCIZIO 5-1:** Si riscriva la subroutine di ritardo utilizzando una tecnica di decremento invece di quella di incremento. Si ricalcolino i numeri da caricare in X ed Y in modo che il ritardo risultante generato dalla subroutine sia approssimativamente 0,1 secondi. Qual è il vantaggio dell'impiego di una tecnica di decremento rispetto a quella di incremento.

**ATTENZIONE:** Se si decide di utilizzare una tecnica di incremento per la generazione del ritardo non si dimentichi di cambiare la locazione 09FC della memoria. Infatti prima di richiamare questa routine occorre caricare una costante diversa.

**ESERCIZIO 5-2:** Si modifichi il programma in modo che il semaforo lampeggi ogni secondo. Inoltre si abbrevi il programma impiegando EOR per commutare il semaforo da una configurazione ad un'altra.

## Modo giorno

In questo modo ogni semaforo commuta tra verde, giallo e rosso con una sequenza consueta. Mentre nella direzione A c'è verde o giallo, nella direzione B è rosso e viceversa. La Figura 5-13 riporta il diagramma di flusso corrispondente all'algoritmo di controllo. Le frecce sulla destra del diagramma di flusso indicano la durata di accensione di ogni luce. Se si chiama D1 la durata del verde e D2 la durata del giallo per la direzione A, D3 e D4 rispettivamente le durate del verde e del giallo per B, si può vedere dall'analisi del diagramma, che la durata totale di un ciclo è  $D1 + D2 + D3 + D4$ .

In un incrocio reale questi ritardi sono imposti da vincoli ben precisi. In particolare la durata del ciclo è normalmente compresa tra uno e due minuti. Il massimo è imposto dal fatto che gli automobilisti non possono tollerare una durata del rosso di oltre due minuti in qualsiasi direzione. Inoltre gli altri ritardi sono imposti dal tempo necessario ad un veicolo o ad un pedone per attraversare l'incrocio. Il periodo del giallo è anche denominato tempo di sgombero per indicare il tempo necessario ad un'automobile per sgomberare l'incrocio. Il limite minimo del verde è imposto dalla presenza o meno di pedoni. Se dei pedoni possono attraversare l'incrocio la durata minima del verde deve essere tale che essi possano attraversare con sicurezza l'incrocio. Per esempio, la durata del rosso nella direzione B, è uguale a  $D1 + D2$ . Se si assume che la durata minima del giallo, nella direzione A sia di 3 secondi, si può vedere dalla Figura 5-13 che la durata minima del rosso nella direzione B è 10 secondi e quindi che la durata minima del verde nella direzione A è  $D1 = 10 - 3 = 7$  secondi. Matematicamente:

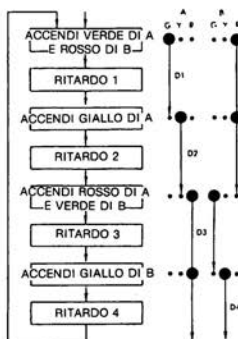


Figura 5-13: Modo Giorno (i comandi non indicati sono spenti)



Se si pone:

VERDE A=D1  
GIALLO A=D2  
VERDE B=D3  
GIALLO B=D4

Allora:

ROSSO A=D3+D4  
ROSSO B=D1+D2

In generale il ciclo è fissato e  $D1 + D2 + D3 + D4 = \text{COSTANTE}$ .

Nel nostro programma si utilizzeranno cicli più veloci di quelli reali. Questo semplicemente perché è spiacevole aspettare uno o due minuti per osservare il funzionamento corretto del semaforo. Per scopi pratici è desiderabile avere un ciclo della durata di  $10 + 20$  secondi, così da controllarlo in modo comodo. Il lettore dovrebbe avere acquisito tutti gli artifici per regolare facilmente il ritardo in modo che il suo microcomputer possa essere connesso ad un incrocio reale. La Figura 5-14 riporta il programma.

0140	A9	3F		DAY	LDA	#\$3F	
0142	8D	03	A0		STA	\$A003	Poni VIA #1 DDRA = \$3F nel modo di uscita
0145	A9	21		ONDAY	LDA	#\$21	
0147	8D	01	A0		STA	\$A001	Accendi il verde ed il rosso in due direzioni
014A	A9	00			LDA	#\$D0	
014C	85	00			STA	\$00	Poni conteggio di DLYA = \$D0 alla locazione \$0000
014E	20	20	01		JSR	DLYA	Richiama ritardo
0151	A9	22			LDA	#\$22	Accendi giallo e rosso
0153	8D	01	A0		STA	\$A001	
0156	A9	EA			LDA	#\$EA	
0158	85	00			STA	\$00	Poni conteggio di DLYA = \$EA
015A	20	20	01		JSR	DLYA	Richiama ritardo
015D	A9	0C			LDA	#\$0C	Accendi rosso e verde
015F	8D	01	A0		STA	\$A001	
0162	A9	00			LDA	#\$D0	
0164	85	00			STA	\$00	Poni indice di DLYA = \$D0
0166	20	20	01		JSR	DLYA	Richiama ritardo
0169	A9	14			LDA	#\$14	Accendi rosso e giallo
016B	8D	01	A0		STA	\$A001	
016E	A9	E8			LDA	#\$E8	
0170	85	00			STA	\$00	Poni indice di DLYA = \$E8
0172	20	20	01		JSR	DLYA	Richiama ritardo
0175	4C	45	01		JMP	ONDAY	Ripeti

Figura 5-14: (Programma 5-2): Simulazione del Semaforo: Modo Giorno (Connessione: connettore A al connettore H1)

Come nel programma precedente, il Registro di Direzione Dati DDRA deve essere configurato nel modo di uscita per il controllo dei 6 LED ad esso connessi. Questo viene realizzato dalle prime due istruzioni del programma:

```
DAY      LDA      #$3F
          STA      $A003
```

Quindi le due istruzioni successive accendono il verde per la direzione A ed il rosso per la direzione B, caricando il bit pattern corretto (21 esadecimale) nel registro di I/O:

```
ON DAY   LDA      #$21
          STA      $A001
```

La durata del ritardo viene quindi specificata caricando un valore nella locazione di memoria 00 e richiamando la subroutine di ritardo:

```
LDA    #$D0
STA    $00
JSR    DLYA
```

Il processo viene quindi ripetuto per il giallo nella direzione A, il rosso nella direzione A, il verde nella direzione B ed, infine, il giallo nella direzione B, per poi tornare al punto di partenza:

```
LDA    #$22      GIALLO A E ROSSO B
STA    $A001
LDA    #$EA
STA    $00
JSR    DLYA      RITARDO
LDA    #$0C      ROSSO A E VERDE B
STA    $A001
LDA    #$D0
STA    $00
JSR    DLYA      RITARDO
LDA    #$14      ROSSO A E GIALLO B
STA    $A001
LDA    #$E8
STA    $00
JSR    DLYA      RITARDO
JMP    ONDAY     RIPETI
```

Il lettore dovrebbe verificare che il programma corrisponde esattamente al diagramma di flusso di Figura 5-13. A questo punto la sua comprensione dovrebbe essere diretta. Si incoraggia vivamente il lettore a ricavare le diverse costanti di tempo utilizzate nel programma ed a verificare che la temporizzazione sia corretta. Consideriamo ora dei miglioramenti a questo algoritmo di controllo del traffico.

Per esempio, si può modificare il programma in modo che la durata del giallo, del rosso e del ciclo sia determinata dal tempo in cui viene premuto uno degli interruttori dopo l'inizio del programma.

**ESERCIZIO 5-3:** Si realizzi un "algoritmo di risposta dinamico": il periodo del verde per l'arteria A sarà esteso di cinque secondi ogni volta che si rivela una richiesta sul "rivelatore ad anello" (o interruttore), fino ad una durata massima del verde di tre minuti.

**ESERCIZIO 5-4:** Si realizzi la "chiamata per i pedoni" impiegando degli interruttori. Si deve fornire il verde ai pedoni prima possibile rispettando un minuto di sgombero.

**ESERCIZIO 5-5:** Si realizzi un "interruttore per la polizia": premendo un interruttore, l'incrocio verrà regolato manualmente. Premendolo due volte consecutive l'incrocio ritorna nel funzionamento automatico.

## LED A MATRICE DI PUNTI

Si utilizzerà un display LED a matrice di 5 x 7 punti (Vedere Figura 5-15). Questo tipo di matrice viene utilizzato in numerose applicazioni. Per esempio, le stampanti a matrice di punti impiegano spesso una matrice 5 x 7 per la stampa di caratteri su carta. I monitor TV od i display CRT impiegano anch'essi una matrice di punti per mostrare i caratteri sullo schermo. 5 x 7 è la matrice di punti minima standard per una rappresentazione accettabile dei caratteri ma non è la migliore in termini di leggibilità. Per migliorare la leggibilità vengono utilizzate matrici di punti più grandi, come 7 x 9, che però hanno un costo più elevato. In questa applicazione si colle-

gerà direttamente una matrice LED di 5×7 punti al registro di I/O B del 6522 #1 e del 6522 #3. In realtà con i LED si dovrebbero utilizzare dei driver per fornire un'intensità luminosa sufficiente. In questa sede per minimizzare i componenti, si conatterà direttamente il LED. Questo significa che sulla scheda reale i LED saranno scuri e difficili da vedere. Per migliorare questa situazione occorre aggiungere i driver sulle linee. La Figura 5-16 mostra la connessione alla matrice di punti LED. Le 7 righe, numerate da 1 a 7 sono connesse rispettivamente ai bit 7, 5, 4, 3, 2, 1 e 0 del registro di I/O B del 6522 #1. Il bit 6 di questo IORB non è disponibile sulla scheda SYM in quanto il monitor dedica il bit 6 alla funzione d'ingresso della cassetta. Quindi, in questo caso, lo stato del bit 6 sarà indifferente.

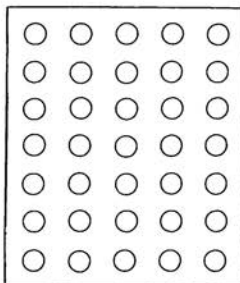


Figura 5-15: Una matrice LED di 5 × 7 Punti

Le cinque colonne del display LED, contrassegnate rispettivamente da 1 a 5, sono connesse ai bit da 0 a 4 di IORB del 6522 #3. Questo è mostrato in Figura 5-16. I due IORB risiedono rispettivamente agli indirizzi A000 ed AC00.

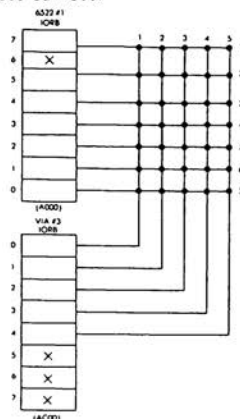


Figura 5-16: Connessione dei LED 5 × 7

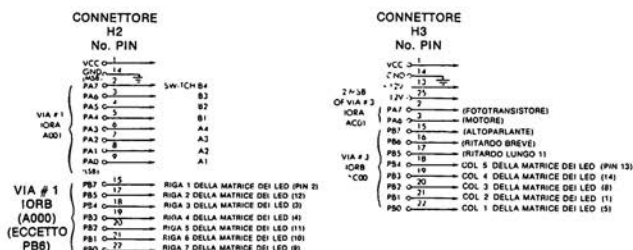


Figura 5-17: I connettori ai LED

Il problema principale è quello di selezionare le combinazioni corrette di righe e di colonne per la rappresentazione dei caratteri. Una matrice di punti  $5 \times 7$  può rappresentare qualsiasi carattere dell'alfabeto. Per esempio si possono mostrare tutti i caratteri esadecimali, cioè le cifre da 0 a 9 e le lettere dalla A alla F. Consideriamo la loro codifica.

Un punto LED acceso sarà rappresentato da uno "0". Un punto LED spento sarà rappresentato da un bit "1". Questo deriva dal fatto che un LED viene acceso collegando a massa la sua connessione di riga. La Figura 5-18 mostra il pattern (struttura) richiesto per rappresentare uno "0". Naturalmente si possono scegliere dei pattern alternativi per qualsiasi altro carattere che si vuole rappresentare. Per esempio, per esercizio, l'utente può utilizzare uno "0" con angoli quadrati piuttosto che rotondi. È molto semplice modificare conseguentemente la tabella.

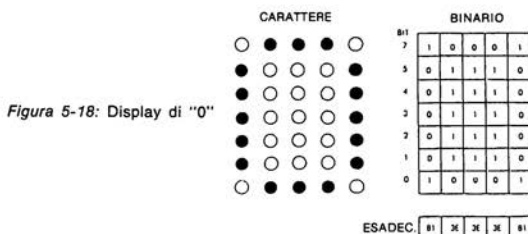


Figura 5-18: Display di "0"

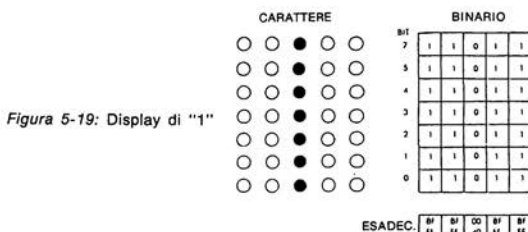


Figura 5-19: Display di "1"

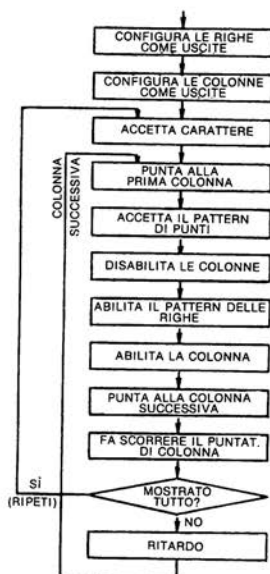


Figura 5-20: Comando dei LED a Matrice di Punti

La rappresentazione binaria equivalente della codifica è riportata sulla destra della Figura 5-18. In fondo alla tabella binaria viene indicato l'equivalente esadecimale. Si ricorda che la riga 6 non viene utilizzata e quindi il suo stato è indifferente. Per esempio si osservi la codifica esadecimale per il carattere "0" in Figura 5-18. La prima colonna ha il valore "1000001" o, più precisamente "1-000001", dove "-" rappresenta il valore del bit 6, che non viene utilizzato. Per esempio assumendo che il bit 6 sia posto al valore "0" il valore della prima colonna è "10000001" cioè "81" in esadecimale.

Analogamente il valore della seconda colonna è (aggiungendo uno 0 per il bit 6) "00111110" ovvero "3E" esadecimale.

Le cinque colonne per la cifra "0" sono perciò:

81, 3E, 3E, 3E, 81

Consideriamo ora il carattere "1". Esso viene rappresentato nella Figura 5-19 e la codifica binaria richiesta appare sulla destra dell'illustrazione.

Assumendo che il bit 6 sia uno "0" le rappresentazioni esadecimali equivalenti sono:

BF, BF, 00, BF, BF

Se si assume che il bit 6 sia posto al valore 1, allora la codifica sarà:

FF, FF, 40, FF, FF

La Figura 5-21 mostra una tabella completa per la codifica dei caratteri da "0" ad "F".

**ESERCIZIO 5-7:** Indicare la forma dei caratteri da 0 ad F impiegando questa tabella.

**ESERCIZIO 5-8:** Si riscriva la tabella in un modo più consistente assumendo che il bit 6 sia sempre "0".

Il diagramma di flusso per il programma della matrice di punti LED è riportato in Figura 5-21. Sia le righe che le colonne sono configurate come uscite caricando il bit pattern opportuno nei registri di direzione dati corrispondenti del 6522. Si deve quindi mostrare il pattern di punti per il carattere.

I punti saranno mostrati in successione per ogni colonna del LED. Per ogni carattere il programma deve perciò accedere a cinque ingressi successivi della tabella della matrice di punti, in corrispondenza delle cinque colonne dei punti richiesti per mostrare il carattere. Questo programma quindi mostrerà indefinitamente il carattere in modo ciclico. I punti vengono mostrati spegnendo le colonne (cancellando il pattern precedente), quindi abilitando il pattern di riga corrispondente alla posizione richiesta dei punti ed abilitando le colonne che devono essere illuminate. Quindi si passa a mostrare la colonna successiva.

Tutti i punti dovrebbero rimanere accesi per lo stesso tempo, per apparire all'osservatore con un'intensità uniforme. Inoltre tutte le colonne devono essere esplorate in un periodo di tempo minore di un decimo di secondo per evitare effetti di lampeggio. La routine di ritardo alla fine del programma deve essere regolata conseguentemente. Il programma è riportato di seguito ed alla pagina successiva.

carattere	8 LSB indir.	col 1	col 2	col 3	col 4	col 5
0	90	81	3E	3E	3E	81
1	95	FF	FF	00	FF	FF
2	9A	DE	7C	7A	76	CE
3	9F	DD	76	76	76	C9
4	A4	F3	EB	DB	00	FB
5	A9	05	76	76	76	79
6	AE	C1	76	76	76	D9
7	B3	7F	7F	7F	7F	00
8	B8	C9	76	76	76	C9
9	BD	CD	76	76	76	C1
A	C2	E0	DB	7B	DB	E0
B	C7	00	76	76	76	C9
C	CC	C1	7E	7E	7E	DD
D	D1	00	7E	7E	7E	C1
E	D6	00	76	76	76	76
F	DB	00	77	77	77	77

La tabella risiede alle locazioni di memoria da 0090 a 00DF.

**Figura 5-21:** Una tabella per Matrice di Punti

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma accetta gli 8 LSB dell'indirizzo del carattere dalla locazione 0001, quindi entra nella tabella di Figura 5-21 per rivelare il pattern di dati del carattere selezionato e mostrarli sulla matrice dei LED.

Prima dell'esecuzione di questo programma occorre caricare gli 8 LSB dell'indirizzo del carattere alla locazione 0001.

Il pattern del carattere dovrebbe essere memorizzato a Pagina 0, come indicato in Figura 5-21.

(Gli 8 MSB dell'indirizzo del carattere sono tutti 00 in Pagina 0)

- Note:
- 1) Per sostituire la tabella si può utilizzare un generatore di caratteri.
  - 2) La matrice di LED utilizzata è  $5 \times 7$ , cioè sono richiesti 7 bit per definire il pattern di ogni colonna, ma la tabella precedente impiega 8 bit: questo perché il programma utilizza il registro B VIA = 110 per pilotare le 7 righe e quindi si possono utilizzare solo 7 bit di questo registro. Il bit 6 è indifferente perché dedicato solo all'ingresso della scheda delle cassette.

0180	A9	BF		BSCLED	LDA	#\$BF	Prima dell'esecuzione dovrebbe essere pre-impostato 0001
0182	8D	02	A0		STA	\$A002	All'indirizzo del carattere selezionato.
0185	A9	1F			LDA	#\$1F	Poni VIA #1 DDRB = BF per pilotare 7 righe
0187	8D	02	AC		STA	\$AC02	Poni VIA #3 DDRB = 1F per pilotare 5 colonne
018A	A9	00			LDA	#\$00	
018C	85	03			STA	\$03	Poni gli 8 MSB dell'indirizzo del carattere a da 00 a 0003
018E	A2	00			LDX	#\$00	
0190	A5	01		RPTCHA	LDA	\$01	Trasferisci gli 8 LSB pre-imposti dell'indirizzo del carattere da 0001 a 0002
0192	85	02			STA	\$02	
0194	A0	10			LDY	-\$10	Poni (Y) = \$10 per abilitare l'ultima colonna
0196	A1	02		NXTCOL	LDA	\$02	(A) = pattern della colonna corrente del carattere selezionato
0198	8E	00	AC		STX	\$AC00	Disabilita tutte le colonne prima di abilitare le righe
019B	8D	00	A0		STA	\$A000	Abilita le righe
019E	8C	00	AC		STY	\$AC00	Abilita la colonna corrente
01A1	E6	02			INC	\$02	Avanza l'indirizzo in (\$0002) per la colonna successiva
01A3	98				TYA		
01A4	4A				LSR	A	Fa scorrere (Y) a destra di un bit per abilitare la colonna successiva
01A5	A8				TAY		
01A6	C0	00			CPY	#\$00	(Y) = 00 significa che sono state mostrate tutte le 5 colonne
01A8	D0	03			BNE	DLY3	Se no, salta a DLY3 per compensare la temporizzazione (1), se si ripeti l'intero carattere
01AA	4C	90	01		JMP	RPTCHA	
01AD	A2	FF		DLY3	LDX	#\$FF	
01AF	E8			LP3	INX		
01B0	E0	00			CPX	#\$00	
01B2	30	FB			BMI	LP3	
01B4	4C	96	01		JMP	NXTCOL	Quindi vai ad abilitare la colonna successiva

- Note:
- 1) Questa compensazione è necessaria oppure anche l'ultima colonna sarà abilitata più a lungo e quindi A sarà più luminosa delle altre 4.
  - 2) La compensazione precedente risolve il problema solo parzialmente. La luminosità non è ancora uniforme in quanto, per ogni colonna è diverso il numero di LED abilitati. Per risolvere questo problema è necessario quindi un programma più dettagliato che tenga conto del numero di LED abilitati per ogni colonna.

Figura 5-22: Display di Base della Matrice di LED

Le prime quattro istruzioni del programma condizionano i registri di direzione dati per le righe e le colonne, specificando che sono uscite:

```
BSCLED   LDA      #$BF
          STA      $A002    PONI VIA #1 = 7 RIGHE
          LDA      #$1F
          STA      $AC02    PONI VIA #2 = 5 COLONNE
```

In questo programma si assume che la locazione del carattere da mostrare sia contenuta alla locazione di memoria "01" di pagina 0. Per esempio la locazione del carattere da mostrare sia 90 per il carattere "0", "95" per il carattere "1" ecc., come indicato in tabella all'inizio del programma. (In seguito verrà suggerita una versione migliorata del programma).

Per esempio, se si deve mostrare il carattere "2", si deve caricare il valore 9A all'indirizzo di memoria 01. Poiché sarà successivamente necessario puntare a 5 ingressi della tabella per ciascuna delle colonne corrispondenti a questo carattere, sarà necessario generare gli indirizzi 9A, 9B, 9C, 9D e 9E. Per non distruggere l'originario puntatore di carattere "9A" si utilizzeranno altre due locazioni di memoria agli indirizzi 02 e 03 per contenere il puntatore corrente alla colonna da mostrare. Poiché si sta operando in pagina 0, i contenuti della locazione di memoria 03 saranno posti a "0" (byte di ordine elevato dell'indirizzo). Questo si ottiene con:

```
LDA      #$00
STA      $03
```

Ogni volta che si entra nel ciclo di display principale, si assume che il registro X contenga il valore "00". Per disabilitare un registro d'uscita si utilizzerà:

```
LDX      #$00
```

La prima colonna a cui si punterà sarà quella di indirizzo specificato dalla locazione 01 (il puntatore all'ingresso della tabella dei caratteri). Quindi si trasferiranno i contenuti della locazione di memoria all'indirizzo 02:

```
RPTCHA   LDA      $01
          STA      $02
```

Il registro Y viene utilizzato come contatore di shift e contemporaneamente, per abilitare selettivamente una delle colonne:

```
LDY      #$10
```

Quindi l'"1" verrà fatto scorrere a destra di una posizione di bit, in modo da abilitare la colonna successiva e così via. Infine, quando l'"1" esce dal registro, sono state mostrate tutte le cinque colonne del carattere e si può riprendere il ciclo. Poiché questo registro non viene utilizzato solo per abilitare una delle cinque colonne ma anche per contare fino a 5, esso viene denominato *shift-counter*. Il pattern di punti per la colonna corrente viene ottenuto mediante l'accesso all'ingresso della tabella all'indirizzo 02:

```
NXTCOL   LDA      $02
```

Il pattern di punti è ora contenuto nell'accumulatore. Per mostrarlo tutte le colonne vengono prima disabilitate caricando "0" in IORB:

```
STX      $AC00
```



Quindi i contenuti dell'accumulatore vengono fatti uscire in IORB per abilitare le righe:

STA            \$A000

Infine viene abilitata la colonna corretta e verrà acceso il LED selezionato:

STY            \$AC00

Un LED si accenderà solo se connesso ad una colonna attiva e ad una riga collegata a massa (0). Ogni "0" nel pattern di punti illuminerà il punto corrispondente nella colonna selezionata.

Viene quindi incrementata la locazione di memoria "02", in modo da puntare all'ingresso del pattern di punti successivo del carattere. Occorre quindi fare scorrere il puntatore di colonna a destra di una posizione e determinare se si sono mostrate tutte le colonne:

INC	\$02	
TYA		Y NON PUÒ ESSERE FATTO SCORRERE DIRETTAMENTE
LSR	A	
TAY		RIMEMORIZZA IL RISULTATO IN A
CPY	#\$00	
BNE	DLY3	
JMP	RPTCHA	

Poiché non è possibile fare scorrere direttamente il registro Y, occorre prima trasferirlo nell'accumulatore, dove viene fatto scorrere, quindi i contenuti dell'accumulatore vengono ricopiati nel registro Y. Si controlla successivamente se i contenuti dell'accumulatore sono uguali a "0" (per questa codifica si suggerirà un programma migliore). Se l'accumulatore è "0", sono state mostrate tutte le cinque colonne. Altrimenti occorre realizzare un ritardo durante il quale si illuminerà il LED e quindi mostrare la colonna successiva:

DLY3	LDX	#\$FF
	INX	
	CPX	#\$00
	BMI	LP3
	JMP	NXTCOL

Il registro indice X viene utilizzato come contatore ed un ritardo viene ottenuto, come al solito, decrementando il registro indice un numero ragionevole di volte, quindi ritornando all'indirizzo NXTCOL per la colonna successiva.

*Miglioramenti del programma:* Per migliorare questo programma riducendo il numero di istruzioni, si considerano in primo luogo alcune modifiche di codifica. Si considereranno successivamente i miglioramenti alle funzioni eseguite.

**ESERCIZIO 5-9:** Si riscriva la routine di ritardo DLY3 in modo che essa impieghi un minor numero di istruzioni.

**ESERCIZIO 5-10:** Si esaminino le ultime tre istruzioni della routine NXTCOL, dall'indirizzo 01A6 in poi (Vedere Figura 5-22). Potete suggerire un altro modo per controllare se è uscito l'ultimo bit "1" da Y?

**ESERCIZIO 5-11:** Si aggiunga una routine a questo programma in modo che, invece di depositare un puntatore all'ingresso della tabella all'indirizzo 01, sia necessario depositare solo il valore del carattere corrente. Con questa routine l'utente deve essere in grado di depositare un

valore corrente tra "0" ed "F" ed ottenere che il programma lo mostri correttamente. Per fare questo occorre convertire il valore del carattere nel valore della tabella. Per esempio "0" corrisponderà a "90" (si veda la tabella all'inizio del programma 5-3). "1" corrisponderà a "95" e così via. L'equazione risulta: Indirizzo di partenza = 90 + codice  $\times$  5.

NOTA: Invece di eseguire una moltiplicazione formale per 5 si può procedere come segue: Si ricordi che lo spostamento a sinistra di una posizione di bit è equivalente ad una moltiplicazione per due e che  $5 = 2 + 2 + 1$ . Quindi una moltiplicazione per 4 può essere eseguita con due successivi scorrimenti a sinistra.

ESERCIZIO 5-12: Si scriva una routine addizionale che mostri una stringa di caratteri. Si assuma che l'indirizzo di partenza della stringa di caratteri sia contenuto nella locazione di memoria 01. Ogni carattere dovrà essere mostrato per un secondo. La stringa di caratteri può terminare con un codice qualsiasi che non è compreso tra "0" ed "F". Il programma eseguirà quindi una pausa di due secondi e mostrerà ancora la stringa.

Si considerano i miglioramenti alle funzioni del programma. Si aggiungeranno quattro interruttori e si svilupperà un programma che mostra il valore esadecimale degli interruttori.

## VISUALIZZAZIONE DEL VALORE ESPRESSO DAGLI INTERRUITORI

Si leggeranno i valori binari di quattro interruttori d'ingresso e si mostrerà il corrispondente carattere esadecimale sulla matrice LED. La Figura 5-23 riporta il diagramma di flusso dell'algoritmo. Il programma legge i quattro interruttori e quindi punta all'inizio della tabella di conversione, nel modo definito nel programma precedente, quindi calcola l'offset della tabella per il carattere da mostrare (fuori zero). L'indirizzo della tabella per il codice binario corrispondente al punto da illuminare è ottenuto moltiplicando il valore del carattere per 5. Questo può essere verificato osservando la tabella mostrata in Figura 5-22. Viene quindi calcolato l'indirizzo della prima colonna da mostrare e caricato all'indirizzo 01 in pagina 0. Il programma precedente viene utilizzato per mostrare il carattere sul display LED. Il programma è il seguente:



Figura 5-23: Display di Valori Impostati con Interruttori

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma legge gli switch da A1 ad A4 per calcolare uno dei 16 valori esadecimali e mostrarlo.

Questo programma impiega il programma 5-3 come subroutine. Prima dell'esecuzione si cambi il programma 5-3 come segue:

- 1) Alla locazione 01A8, i dati 4C dovrebbero essere sostituiti con 60 (60 è il codice di macchina per RST).
- 2) La costante di compensazione temporale della locazione 1AC è FF. questo valore dovrebbe essere cambiato in F0, questo perché questo programma abilita l'ultima colonna per un tempo più lungo del programma 5-3.

0200	A9	00		RDCHA	LDA	#\$00	
0202	8D	03	A0		STA	\$A003	Poni VIA #1 DDRA = 00 nel modo d'ingresso
0205	AD	01	A0		LDA	\$A001	Leggi gli switch B1-B4 ed A1-A4
0208	29	0F			AND	#\$0F	Ignora B1-B4
020A	A8				TAY		Memorizza A1-A4 leggendoli in (Y)
020B	A2	90			LDX	#\$90	Calcola l'indirizzo del carattere e memorizzato alla locaz. 0001 La base dell'indir. è 90
020D	86	01			STX	\$01	
020F	A2	00			LDX	#\$00	Il contatore dell'addizione
0211	18			ADD	CLC		A contiene la lettura degli switch
0212	65	01			ADC	\$01	Ricicla attraverso l'addizione 5 volte
0214	85	01			STA	\$01	90 + (A)
0216	98				TYA		
0217	E8				INX		Rimemorizza il valore degli switch in A.
0218	E0	05			CPX	#\$05	(X) = 5 significa calcolo terminato
021A	30	F5			BMI	ADD	
021C	20	80	01		JSR	BSCLED	Quindi chiama BSCLED per display
021F	4C	00	02		JMP	RDCHA	Quindi aggiorna la lettura degli switch

Figura 5-24: Display Avanzato a Matrice di LED (Programma 5-4)

Il programma è riportato in Figura 5-24. Le prime due istruzioni configurano come ingresso il registro di direzione dati per la porta A, così da poter leggere gli interruttori:

```
RDCHA    LDA    #$00
          STA    $A003
```

Quindi vengono letti i contenuti degli interruttori da A1 ad A4. Questo programma ignora i valori degli interruttori da B1 a B4.

```
          LDA    $A001
          AND    #$0F    MASCHERA B1-B4
```

I contenuti indicati dagli interruttori vengono caricati nel registro indice Y:

```
TAY
```

L'indirizzo di partenza della tabella (90) viene quindi memorizzato all'indirizzo di memoria 01:

```
          LDX    #$90
          STX    $01
```

A questo indirizzo di partenza si aggiungerà l'offset richiesto per accedere alla prima colonna di punti per il carattere specificato dagli interruttori. L'offset viene calcolato moltiplicando il valore degli interruttori per 5. Il registro indice X viene utilizzato come contatore da 0 a 5. Esso viene inizializzato a zero:

```
          LDX    #$00
```

I contenuti della locazione di memoria 01 vengono incrementati di 1:

ADD	CLC	
	ADC	\$01
	STA	\$01

L'istruzione CLC (azzerà carry) deve essere impiegata prima di un'addizione qualsiasi. Nell'addizione si assume che sia stato scelto il modo binario (il 6502 può funzionare nel modo decimale oppure nel modo binario). Salvo diversa specifica, normalmente il 6502 funzionerà nel modo binario, poiché un'operazione di reset avrà azzerato tutti i registri dei flag, impostando quindi il modo binario.

Il valore degli interruttori viene quindi rimemorizzato nell'accumulatore dal registro Y dove era stato conservato. Il contatore dell'addizione X viene incrementato di 1 e controllato se uguale a 5:

TYA	
INX	
CPX	#\$5
BMI	ADD

L'addizione viene ripetuta finché non si raggiunge il valore 5. Una volta raggiunto questo valore, la locazione di memoria 01 viene condizionata al valore corretto e viene richiamata la subroutine BSCLED (il precedente programma del display LED):

JSR	BSCLED
-----	--------

Quindi il programma ritorna indietro per leggere ancora gli interruttori e mostrare il carattere che esprimono:

JMP	RDCHA
-----	-------

## GENERAZIONE DI TONO

Nel capitolo precedente si è mostrato come si può generare un tono inviando semplicemente un'onda quadra, di una certa frequenza, ad un altoparlante. La forma d'onda quadra viene generata accendendo e spegnendo alternativamente l'altoparlante. La durata durante la quale l'altoparlante è acceso o spento si chiama semiperiodo. La misura del ritardo può essere eseguita per via software, od anche per via hardware, impiegando un timer di intervallo incorporato nel 6522. Questo timer di intervallo incorporato è stato impiegato precedentemente e qui verrà utilizzato in un metodo *software* per controllare la durata del ritardo. Innanzi tutto si svilupperà un programma di base per generare un tono e quindi lo si migliorerà per generare della musica con il computer.

La Figura 5-25 mostra la connessione hardware. Un resistore addizionale maggiore o uguale a 50 Ohm viene posto in serie all'altoparlante per limitare la corrente d'uscita. L'altoparlante viene connesso ad un'uscita buffered del SYM. Portando a zero il resistore variabile si potrebbe bruciare il potenziometro od il transistor di uscita sul carico.

La tecnica per generare un tono si basa su un comune metodo ad onda quadra, ottenuta mediante una subroutine di ritardo.

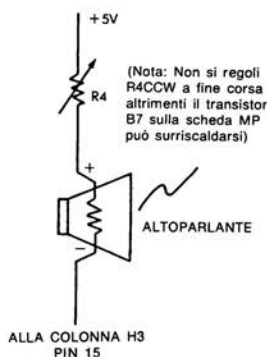


Figura 5-25: Connessione dell'Altoparlante

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma attiva l'altoparlante con una frequenza pre-impostata che deve essere caricata nella locazione 0004 prima dell'esecuzione.

0230	A9	80		BSCSPK	LDA	#S80	
0232	8D	02	AC		STA	\$AC02	Poni VIA #3 DDRB = 80 per l'uscita dell'altoparlante
0235	A9	80		AGAIN	LDA	#S80	
0237	8D	00	AC		STA	\$AC00	Poni alto il driver dell'altoparlante = altoparlante attivato
023A	20	48	02		JSR	DLYB	Richiama ritardo
023D	A9	00			LDA	#S00	
023F	8D	00	AC		STA	\$AC00	Poni basso il driver dell'altoparlante = altoparlante disattivato
0242	20	48	02		JSR	DLYB	Richiama ritardo
0245	4C	30	02		JMP	AGAIN	Ripeti

Subroutine DLYB: questa subroutine è simile alla DLYA tranne:

- 1) Questo ritardo è molto più breve.
- 2) Questo ritardo prende l'indice di ritardo dalla locazione 0004 (l'indice dovrebbe avere un valore negativo).

0248	A6	04		DLYB	LDX	\$04	Carica il valore del ritardo in X
024A	E8			LPXB	INX		Incrementa X
024B	E0	00			CPX	#S00	
024D	30	FB			BMI	LPXB	Ricicla finché (X) = 0
024F	60				RTS		

Figura 5-26: Attivazione di base dell'Altoparlante (Programma 5-5)

Il parametro di ritardo per questo programma deve essere caricato alla locazione di memoria 0004 prima dell'esecuzione. Esso controlla la frequenza del tono che viene generato. La Figura 5-26 mostra il programma. Il registro di direzione dati B è configurato per l'uscita sul bit 7:

```
BSCSPK    LDA    #$80
           STA    $AC02
```

L'altoparlante viene quindi acceso:

```
AGAIN     LDA    #$80
           STA    $AC00
```

L'altoparlante viene lasciato acceso per una durata imposta dai contenuti della locazione di memoria 0004, richiamando la subroutine di ritardo DLYB:

```
JSR       DLYB
```

Occorre quindi spegnere l'altoparlante. Questo viene eseguito ripristinando a "0" il bit 7 di IORB:

```
LDA       #$00
STA       $AC00
```

L'altoparlante deve quindi essere lasciato spento per la stessa durata e questo viene nuovamente realizzato con una chiamata alla subroutine DLYB:

```
JSR       DLYB
```

Il programma quindi ricicla su se stesso:

```
JMP       AGAIN
```

La routine di ritardo DLYB è essenzialmente analoga alla DLYA del programma 5-1:

DLYB	LDX	\$0A	VALORE RITARDO
LPXB	INX		CONTATORE
	CPX	#\$00	
	BMI	LPXB	
	RTS		

Calcoliamo la durata del ritardo introdotto da questa subroutine. Di seguito viene indicata la durata di ciascuna istruzione (sulla destra):

				# cicli
	LDX	\$04		(2)
	INX			(2)
	CPX	#\$00		(2)
	BMI	LPXB		(3)
	RTS			(6)
Ciclo				

Inoltre l'istruzione JSR (salta alla subroutine), impiegata per la chiamata di questa subroutine, introduce un ritardo di 6 cicli. Il ciclo viene eseguito  $256 - 4 = 252$  volte.

Quindi la durata di ritardo totale vale perciò:

$$6 + 2 + (2 + 2 + 3) \times 252 + 6 = 14 + 7 \times 252 = 1778 \text{ microsecondi}$$

**ESERCIZIO 5-13:** Si modifichi la routine di ritardo impiegando un'istruzione di decremento invece di una di incremento.

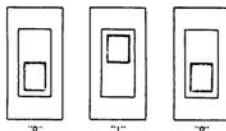


Figura 5-27: Gli Switch Binari Specificano il Tono

## MUSICA

È stato presentato un metodo di base per la generazione di un tono di una frequenza assegnata. Vogliamo realizzare ora un accordo. Questo programma leggerà il valore binario dei tre interruttori da A1 ad A3 e genererà un tono corrispondente al posizionamento dell'interruttore (vedere Figura 5-27). Per il posizionamento "0" dell'interruttore verrà generata una nota "C" (do), quindi un "D" (re) per un "1", ecc. Un'ottava intera più una nota, cioè da "C" a "C", possono essere accordate posizionando tre interruttori. Questo programma impiegherà quello precedente come subroutine. Prima dell'esecuzione del programma i contenuti della locazione di memoria 0245 dovrebbero essere cambiati da "4C" a "60". Innanzi tutto si costituirà una tabella di frequenze, che specifica la durata del semiperiodo dell'onda quadra che genera il tono. La Figura 5-28 riporta tale tabella.

0050	A2	80		TUNE	LDX	#S80	Frequenza per C intermedio
0052	4C	74	02		JMP	LD04	
0055	A2	90			LDX	#S90	Frequenza per D
0057	4C	74	02		JMP	LD04	
005A	A2	9C			LDX	#S9C	Frequenza per E
005C	4C	74	02		JMP	LD04	
005F	A2	A4			LDX	#SA4	Frequenza per F
0061	4C	74	02		JMP	LD04	
0064	A2	80			LDX	#S80	Frequenza per G
0066	4C	74	02		JMP	LD04	
0069	A2	88			LDX	#S88	Frequenza per A
006B	4C	74	02		JMP	LD04	
006E	A2	C0			LDX	#SC0	Frequenza per B
0070	4C	74	02		JMP	LD04	
0073	A2	C4			LDX	#SC4	Frequenza per C
0075	4C	74	02		JMP	LD04	

Figura 5-28: Tabella delle Frequenze Musicali

La Figura 5-29 riporta il flow-chart dell'algoritmo. Il programma legge i contenuti dei tre interruttori, calcola l'offset richiesto per ottenere il ritardo corrispondente dalla tabella delle frequenze. Questo fuori zero è uguale a cinque volte il valore specificato dagli interruttori. Si ottiene quindi il periodo dell'onda quadra che viene emessa per una durata specifica. Il programma quindi cicla su se stesso in modo da emettere la nota successiva. La Figura 5-30 mostra il programma e la Figura 5-31 le connessioni. Le locazioni 04 e 05 verranno utilizzate per un salto



Figura 5-29: Diagramma di flusso del Programma Musicale

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma legge gli switch A1 - A3 ed aziona l'altoparlante ad 8 frequenze diverse definite dagli switch.

Questo programma utilizza il programma #5 come subroutine; prima dell'esecuzione la locazione 0245 dovrebbe essere cambiata da 4C a 60.

Questo programma esegue l'accordo saltando ad una tabella di frequenze. La tabella delle frequenze deve essere caricata come segue prima dell'esecuzione:

0250	A9	00		MUSIC	LDA	#\$00	Pre-carica gli 8 MSB dell'indirizzo di salto indiretto
0252	85	05			STA	\$05	Alla locazione 0005 (= 00 poiché la tabella delle frequenze è in pagina 0)
0254	8D	03	A0	KEY	STA	\$A003	Poni VIA #1 DDRA = 00 per il modo d'ingresso
0257	A0	C0			LDY	#\$C0	(Y) = costante di ritardo per ogni frequenza
0259	AD	01	A0		LDA	#\$A001	Leggi l'impostazione degli switch
025C	29	07			AND	#\$07	Ignora i 5 bit superiori
025E	85	04			STA	\$04	Salva l'impostazione degli switch in \$04
0260	18				CLC		
0261	65	04			ADC	\$04	
0263	65	04			ADC	\$04	
0265	65	04			ADC	\$04	
0267	65	04			ADC	\$04	Calcola l'indirizzo relativo nella tabella delle frequenze
0269	85	04			STA	\$04	
026B	A9	50			LDA	TUNE	Somma l'indirizzo della base della tabella delle frequenze
026D	85	04			ADC	\$04	
026F	85	04			STA	\$04	Memorizza l'indirizzo calcolato (8 LSB) alla locazione 0004
0271	6C	04	00		JMP	(\$0004)	Salto indiretto alla tabella delle frequenze
0274	86	04		LD04	STX	\$04	Accetta la costante corretta della frequenza
0276	20	30	02	CBSPK	JSR	BSCSPK	Chiama BSCSPK per attivare l'altoparlante
0279	88				DEY		
027A	C0	00			CPY	#\$00	Ricicla finché (Y) = 0 prima di rivelare gli switch
027C	D0	F8			BNE	CBSPK	Ancora
027E	4C	57	02		JMP	KEY	Ritorna a rivelare gli switch

Figura 5-30: Il Programma Musicale (Programma 5-6)



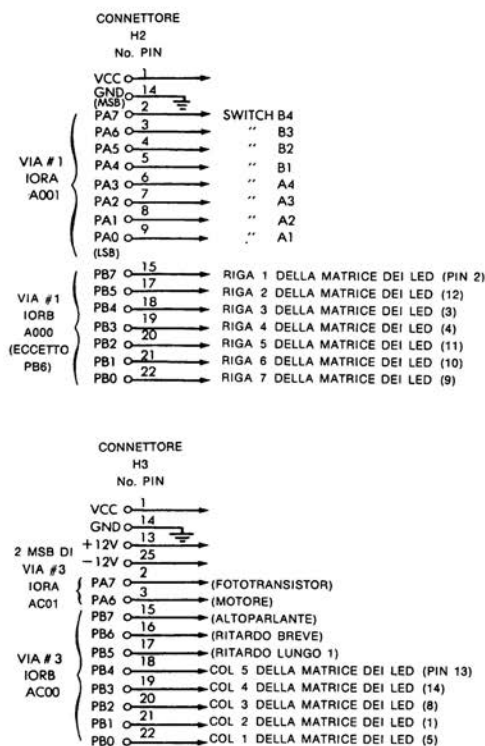


Figura 5-31: Connessioni per il Programma Musicale

indiretto. Poiché la tabella delle frequenze risiede in pagina zero, i contenuti della locazione di memoria 05 saranno immediatamente inizializzati a zero:

```
MUSIC    LDA    # $00
          STA    $05
```

Il registro di direzione dati, DDRA, viene quindi configurato a "00" per specificare il modo d'ingresso:

```
STA      $A003
```

La durata del tono viene specificata dai contenuti del registro Y che corrisponde al ritardo del ciclo più esterno (verrà spiegato in seguito):

```
KEY      LDY      #$C0
```

I contenuti dei tre interruttori A1, A2, ed A3 vengono quindi letti da IORA nella locazione di memoria A001 ed i 5 bit superiori sono mascherati (posti a 0):

```
LDA      #$A001
AND      #$07
```

La posizione degli interruttori viene quindi conservata nella locazione di memoria 04 in modo che l'accumulatore possa essere utilizzato per altri scopi:

```
STA      $04
```

Per calcolare l'offset nella tabella delle frequenze, il valore ottenuto dagli interruttori viene moltiplicato per 5. Questo viene eseguito in questo caso sommando questo valore a se stesso per 4 volte:

```
ADC      $04
ADC      $04
ADC      $04
ADC      $04
STA      $04
```

Il valore di fuori zero risultante viene quindi memorizzato nella locazione di memoria 04 ed ora si può ottenere il semiperiodo dalla tabella delle frequenze:

LDA	TUNE	INDIRIZZO DELLA BASE
ADC	\$04	
STA	\$04	BASE PIÙ SPOSTAMENTO
JMP	(\$0004)	SALTA IN MODO INDIRECTO
STX	\$04	COSTANTE DELLA FREQUENZA

Il valore ritorna nel registro X e salvato nella locazione di memoria 04, quindi viene chiamata la subroutine BSCSPK per attivare l'altoparlante:

```
CBSPK    JSR      BSCSPK
```

L'altoparlante verrà attivato il numero di volte specificato dai contenuti del registro Y:

```
DEY
CPY      #$00
BNE      CBSPK
```

Infine, una volta che il tono è stato generato per una durata specifica, vengono letti ancora gli interruttori:

```
JMP      KEY
```

Si migliori questo programma:

ESERCIZIO 5-14: Si potrebbe semplificare la tabella delle frequenze memorizzando in essa

soltanto il valore binario del ritardo, cioè \$80, \$90, ecc. Si modifichi il programma precedente in modo che il posizionamento degli interruttori venga impiegato come indice per recuperare i contenuti di questa nuova tabella. Si noti il significativo miglioramento in lunghezza del programma globale.

**ESERCIZIO 5-15:** Se si esegue effettivamente questo programma su una scheda di microcomputer si noterà un problema secondario: Il programma esegue la nota richiesta; comunque si può sentire contemporaneamente una nota di frequenza più bassa. Analizzando attentamente le ultime cinque istruzioni del programma per la generazione di musica, si potrebbe determinare l'origine del problema. Siete in grado di proporre un programma modificato che possa eliminare questo problema?

(Suggerimento: L'altoparlante può essere spento "troppo a lungo".)

**ESERCIZIO 5-16:** Si osservi che l'istruzione "ADC \$04" è ripetuta 4 volte, si suggerisca un modo per ottenere lo stesso risultato con il numero minimo di istruzioni.

**ESERCIZIO 5-17:** La terza istruzione dalla fine è "CPY #\$00", è davvero indispensabile?

La tabella del programma musicale è stata progettata "ad orecchio" senza il calcolo delle frequenze corrette. Tali valori dovrebbero ora essere controllati per determinare il grado di bontà di questa tabella.

In America il Passo Standard è  $A_4 = 440$  Hz. La frequenza delle note si raddoppia ogni dodici mezza note. Dal tono  $T_1$  a  $T_2$  la frequenza è  $N_2 = 12 \sqrt{2} \times N_1$ .

Le frequenze sono riportate in Figura 5-28.

**ESERCIZIO 5-18:** Si esamini la routine BSCSPK per calcolare la sua temporizzazione. Conoscendo i periodi delle note (Figura 5-28), si calcolino le costanti di frequenza teoricamente corrette. (Suggerimento: Non si dimentichi che l'altoparlante è alternativamente acceso e spento per un semiperiodo).

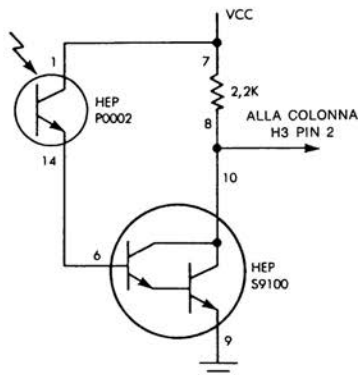


Figura 5-32: Il circuito del Fototransistor (sullo zoccolo M3)

## UN SISTEMA DI ALLARME ANTIFURTO

Vogliamo realizzare un sistema reale di allarme domestico. L'ingresso in casa sarà rivelato da un insieme fototransistor rivelatore. Si assume che l'emettitore di luce sia normalmente acceso. Ogni volta che il raggio luminoso viene interrotto il rivelatore lo indicherà e verrà fatto scattare l'allarme. Questo allarme genererà un suono di sirena per mezzo di un altoparlante. Alla fine del programma si suggeriranno ulteriori miglioramenti.

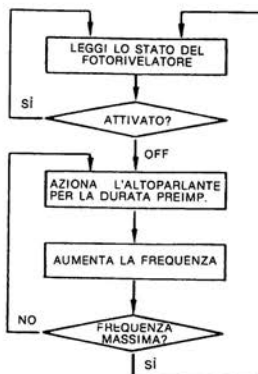


Figura 5-33: Diagramma di flusso del Sistema di Allarme

La Figura 5-32 mostra la connessione del fototransistor, mentre la Figura 5-33 riporta il diagramma di flusso dell'algoritmo. Leggeremo lo stato del rivelatore. Finché esso rimane acceso, nessuno ha interrotto il raggio e si prosegue la lettura. Ogni volta che viene interrotto il raggio lo stato del rivelatore andrà a "0" (spento) e verrà attivato l'altoparlante per una durata prefissata. Per generare un suono di sirena, la frequenza di questo suono andrà progressivamente aumentando fino al raggiungimento di una frequenza massima (Vedere Figura 5-35). A questo punto viene nuovamente esaminato lo stato del fotorelevatore e, finché è spento, la sirena continua a suonare. La Figura 5-34 riporta il programma. L'ingresso del fototransistor è connesso al bit 7 di IORA VIA #3 (Vedere Figura 5-32).

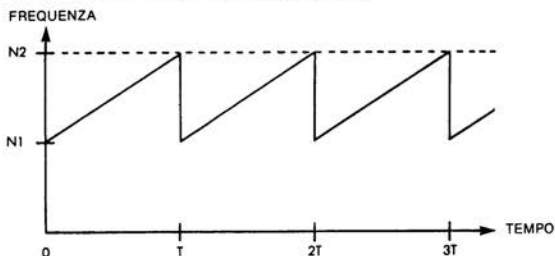


Figura 5-34: Il Suono di una Sirena

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma rivela l'uscita del fototransistor, se essa è alta, cioè se il fototransistor è al buio ed esso è diseccitato, non si opera, invece se l'uscita è bassa, cioè il fototransistor è eccitato da luce, suona immediatamente l'allarme.

Questo programma impiega BSCSPK come subroutine, la locazione 0245 deve essere cambiata in 60.

0281	A9	00		ALARM	LDA	#\$00	
0283	8D	03	AC		STA	\$AC03	Poni VIA #3 DDRA = 00 per il modo d'ingresso
0286	AD	01	AC	DETECT	LDA	\$AC01	Leggi l'uscita del fototransistor
0289	29	80			AND	#\$80	
028B	C9	80			CMP	#\$80	
028D	F0	F7			BEQ	DETECT	Se uscita = alta, continua la registrazione
028F	A9	80			LDA	#\$80	Altrimenti suona l'allarme ponendo la
0291	85	04			STA	\$04	Costante iniziale di frequenza = 80 alla loc. 0004
0293	A0	F0		LP7	LDY	#\$F0	Poni la costante di ritardo = F0 a (Y)
0295	20	30	02	SOUND	JSR	BSCSPK	Chiama BSCSPK per azionare l'altoparlante
0298	C8				INY		
0299	C0	00			CPY	#\$00	
029B	30	F8			BMI	SOUND	Ricicla finché (Y) = 0 prima di cambiare la costante di frequenza
029D	A9	01			LDA	#\$01	Incrementa di 1 la costante di frequenza
029F	18				CLC		
02A0	65	04			ADC	\$04	
02A2	85	04			STA	\$04	
02A4	C9	A8			CMP	#\$A8	Ricicla fino alla costante di frequenza più alta = A8
02A6	30	EB			BMI	LP7	
02A8	4C	86	02		JMP	DETECT	Quindi ritorna di nuovo a rivelare il fototransistor

Figura 5-35: Allarme Antifurto (Programma 5-7)

Le prime istruzioni del programma realizzano un ciclo di polling che controlla lo stato del fototransistor:

ALARM	LDA	#\$00
	STA	\$AC03
DETECT	LDA	\$AC01
	CMP	#\$80
	BEQ	DETECT

Non appena viene spento il fototransistor (su una scheda sperimentale ciò si può ottenere coprendo il rivelatore LED con un dito o un panno), entrerà in funzione l'allarme. Una ben precisa costante di frequenza iniziale viene caricata all'indirizzo di memoria 04 e la durata del tono di questa frequenza viene caricata nel registro Y. La subroutine precedente BSCSPK viene quindi richiamata per far suonare l'altoparlante:

	LDA	#\$80
	STA	\$04
LP7	LDY	#\$F0
SOUND	JSR	BSCSPK
	INY	
	CPY	#\$00
	BMI	SOUND

La subroutine viene richiamata il numero di volte necessario per realizzare il ritardo secondario specificato dal registro Y. Quindi la costante di frequenza viene incrementata di 1, rimemorizzata nella locazione di memoria 04, quindi nuovamente confrontata con la frequenza massi-

ma. Una volta raggiunta la frequenza massima il programma ricomincia la generazione di un suono a frequenza crescente.

LDA	#\$01
CLC	
ADC	\$04
STA	\$04
CMP	#\$A8
BMI	LP7
JMP	DETECT

Ogni volta che viene raggiunta la frequenza massima il programma ritorna al punto di partenza. Sono possibili diversi miglioramenti.

In una casa questo sistema di allarme sarà posizionato in corrispondenza di ogni potenziale ingresso. In ognuno dei quali occorrerà posizionare una coppia fotoelettrica comprendente un emettitore di luce ed un ricevitore. (In pratica si impiega sempre un raggio infrarosso perché invisibile). Si possono posizionare questi sensori per proteggere una stanza o l'ingresso della casa. Il programma potrebbe essere migliorato in modo che, una volta acceso il sistema, sia possibile lasciare la casa senza mettere in funzione l'allarme. Il primo esercizio consisterà proprio in questo miglioramento:

**ESERCIZIO 5-19:** *Si modifichi il programma in modo che sia possibile uscire dalla casa entro due minuti dopo che il sistema è stato attivato. In altre parole non deve scattare nessun allarme per due minuti dopo che il programma è stato attivato, indipendentemente dallo stato del fotorivelatore. Dopo questo tempo l'allarme deve riprendere a funzionare normalmente.*

Occorre risolvere un altro problema: Rientrando in casa non si vuole che l'allarme suoni immediatamente. Si vuole avere il tempo per spegnere il sistema. Il prossimo esercizio consiste in questo:

**ESERCIZIO 5-20:** *Una volta che l'allarme è stato inserito (dopo due minuti) esso non dovrebbe suonare per 30 secondi dopo la rivelazione di un ingresso.*

Ulteriore miglioramento: Potrebbero esserci variazioni secondarie del raggio luminoso a causa del rumore sulla linea. Si vuole evitare che questo faccia scattare l'allarme.

**ESERCIZIO 5-21:** *Si modifichi il programma in modo che sia fatto scattare l'allarme solo se il raggio viene interrotto per più di 0,05 secondi.*

Miglioramento: nel caso che un animale faccia scattare l'allarme si vuole introdurre un sistema automatico di spegnimento. Si vuole che l'allarme suoni per due minuti dopo la rivelazione e che si spenga automaticamente.

**ESERCIZIO 5-22:** *Si modifichi il programma in modo che l'allarme suoni per due minuti dopo che si è verificata una rivelazione e che quindi si spenga automaticamente.*

Inoltre, durante la rivelazione, si possono prendere degli ulteriori provvedimenti quali illuminare l'ambiente oppure telefonare alla polizia. Questo può essere realizzato facilmente mediante un relè esterno.

**ESERCIZIO 5-23:** *Si modifichi il programma precedente in modo che si alimenti un relè esterno ogni volta che si rivela un ingresso.*

Si noti che questa possibilità può essere utilizzata vantaggiosamente anche se non si può telefonare automaticamente alla polizia: si può connettere una lampada al relè per rivelare se c'è stato un allarme.

ESERCIZIO 5-24: Sarebbe possibile eliminare l'istruzione CPY # \$ 00 all'indirizzo 0299?

ESERCIZIO 5-25: Si aggiunga un pulsante di chiamata in caso di paura, che sia possibile attivare in qualsiasi momento. Si modifichi il suono dell'allarme in modo che i vicini possano differenziare tra questa chiamata e l'allarme vero e proprio.

## CONTROLLO DI UN MOTORE IN CORRENTE CONTINUA

Questo programma si propone di controllare la velocità di un motore in corrente continua. Un comune motore di basso costo, per impieghi hobbystici, in c.c. a 12 V può essere connesso alla scheda microcomputer e la velocità di rotazione può essere impostata mediante degli interruttori. Si utilizzeranno tre interruttori, in modo da ottenere otto diverse combinazioni, corrispondenti ad otto velocità di rotazione diverse. La Figura 5-36 mostra il circuito del motore, mentre la Figura 5-27 riporta lo schema delle connessioni degli interruttori.

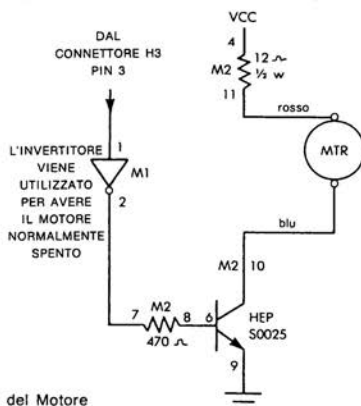


Figura 5-36: Circuito del Motore

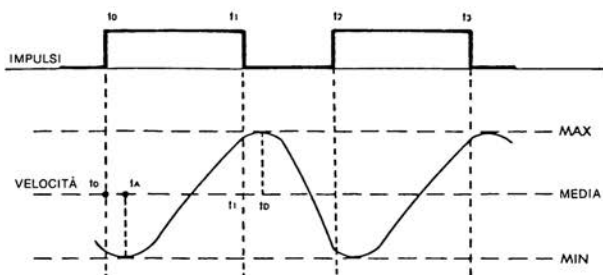


Figura 5-37: Controllo Digitale della Velocità

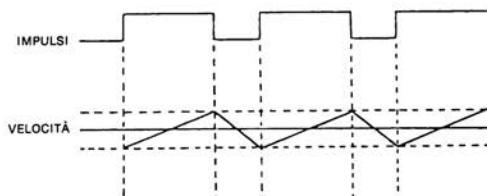


Figura 5-38: Diagramma Semplificato della Velocità

Il principio impiegato per controllare la velocità del motore è di accenderlo per una durata prefissata e quindi di spegnerlo. A causa della sua inerzia rotativa il motore si manterrà in rotazione per un certo tempo. Successivamente verrà generato un nuovo impulso di alimentazione che lo accelererà. Questo schema viene quindi ripetuto. La Figura 5-37 riporta l'andamento della velocità risultante del motore, mentre la Figura 5-38 mostra un diagramma semplificato della stessa curva. Sostanzialmente si tratta di una curva a dente di sega in cui il motore accelera finché si fornisce potenza e decelera quando non è alimentato. Nella Figura 5-37 la velocità media è indicata con una retta orizzontale tra le velocità minima e massima. Dalla figura si può vedere che la velocità oscillerà continuamente tra i valori minimo e massimo. La velocità deve essere definita con buona precisione, quindi i valori massimo e minimo devono essere vicini. Questo si può ottenere impiegando degli impulsi brevi. Comunque, come per qualsiasi fenomeno che coinvolge inerzia ed oscillazioni, si possono verificare delle instabilità. In particolare occorre notare che, se l'impulso di accensione viene fornito prima del tempo " $t_D$ " la

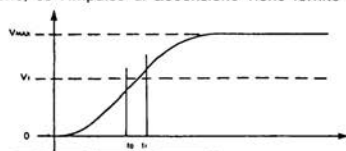


Figura 5-39: Curva di Velocità del Motore in c.c.

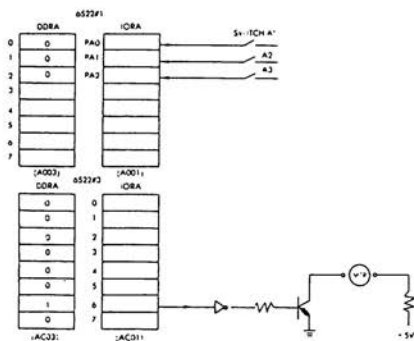


Figura 5-40: Le Connessioni



velocità, invece di decrescere, aumenterà. Si possono verificare inoltre dei fenomeni più complessi, cosa che non verrà fatta in questa sede. Qui si progetterà semplicemente un programma a ritardo regolabile e quindi si regoleranno questi ritardi compatibilmente con il tipo di motore che si sta utilizzando. Si sottolinea che questi ritardi possono essere utilizzati in vari modi per migliorare la precisione della velocità ottenuta e/o eliminare problemi di oscillazioni.

## Le Connessioni Hardware

Si impiegano due porte: 6522 #1 e 6522 #3, come mostrato in Figura 5-40. Il registro IORA viene impiegato come porta d'ingresso per i tre interruttori. Il posizionamento degli interruttori determinerà la velocità del motore. Il valore corrispondente di DDRA è mostrato sulla sinistra della figura. L'ORA del 6522 #3 viene impiegato come porta di controllo per il motore stesso. Il motore è connesso al bit 6 di IORA. La Figura 5-36 riporta i dettagli dell'interfaccia. Il driver è richiesto per invertire il segnale ed il transistor viene impiegato per fornire una corrente sufficiente.

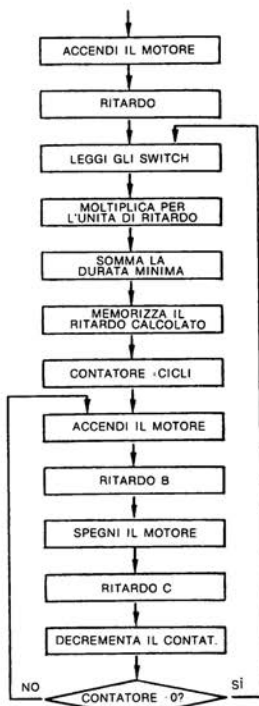


Figura 5-41: Diagramma di flusso del Motore in c.c.

## II Programma

La Figura 5-41 mostra il diagramma di flusso del programma. Il motore rimarrà acceso per un periodo  $T_{on}$  e spento per un periodo  $T_{off}$ . In questo algoritmo la durata di spegnimento è fissa mentre  $T_{on}$  aumenta proporzionalmente al posizionamento degli interruttori da "000" ad "111". In questo caso la durata minima di  $T_{on}$  corrisponde al posizionamento "000". Il ritardo corrispondente ad un certo posizionamento degli interruttori può essere calcolato utilizzando la formula:

$$T_{on} = MIN + Unità \times interruttori.$$

Numericamente, le costanti impiegate per i ritardi sono:

$$\begin{aligned} RITARDO_{OFF} &= COH = 192 \text{ decimale} \\ RITARDO_{ON} &= 80H + interruttori \times 0BH = \\ &= 128 + interruttori \times 11 \text{ (decimale)} \end{aligned}$$

000	001	010	011	100	101	110	111
128	139	150	161	172	183	197	205

INTERRUTTORI	000	001	010	011	100	101	110	111
RITARDO <sub>ON</sub>	128	139	150	161	172	183	197	205

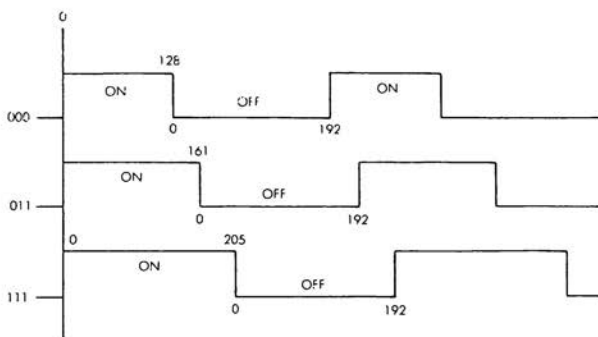


Figura 5-42: Le Forme d'Onda

La Figura 5-42 riporta le forme d'onda generate in corrispondenza delle diverse impostazioni. Si osservi il flow-chart di Figura 5-41: il motore viene acceso per una durata iniziale per ottenere una velocità rotativa iniziale (altrimenti un treno di impulsi potrebbe non essere in grado di farlo partire). Viene quindi letto il valore impostato sugli interruttori e calcolato il ritardo corrispondente. Il valore impostato sugli interruttori, moltiplicato per l'unità di ritardo, viene sommato alla durata minima d'impulso. Il ritardo risultante viene memorizzato. Il motore viene quindi acceso per il ritardo calcolato. Questo è il Ritardo B. Quindi il motore viene spento per una durata chiamata Ritardo C. Questo processo viene ripetuto per diversi cicli per stabilizzare

la velocità. Quindi si eseguirà una nuova lettura degli interruttori e, se l'impostazione è stata cambiata, si genererà una nuova velocità. Si noti che il ritardo generato dalla ripetizione diverse volte del ciclo elimina il problema del rimbalzo. Se non è consentito un ritardo per la stabilizzazione della velocità, si dovrebbe eliminare il rimbalzo degli interruttori per via hardware o per via software (Vedere per i dettagli sull'eliminazione del rimbalzo "Tecniche di interfacciamento dei microprocessori").

Connessione: Connettore A al connettore H2  
Connettore AA al connettore H3

Questo programma legge gli switch A1-A3 per definire la velocità del motore richiesta ed aziona conseguentemente il motore.

Questo programma impiega due subroutine: DLYA e DLYB.

02B0	A9	40		MOTOR	LDA	#S40	
02B2	8D	03	AC		STA	\$AC03	Poni VIA #3 DDRA = 40 per l'uscita del driver del motore
02B5	A9	00			LDA	#S00	Accendi il motore per una durata DLYA per ottenere la velocità iniziale
02B7	8D	01	AC		STA	\$AC01	
02BA	A9	FF			LDA	#SFF	
02BC	85	00			STA	\$00	
02BE	20	20	01		JSR	DLYA	
02C1	A9	00			LDA	#S00	Poni VIA #1 DDRA = 00 per il modo d'ingresso
02C3	8D	03	A0		STA	\$A003	
02C6	AD	01	A0	MTRSP	LDA	\$A001	Leggi gli switch
02C9	29	07			AND	#S07	Ignora i 5 bit superiori
02CB	A8				TAY		(Y) = lettura degli switch
02CC	A9	0B			LDA	#S0B	Poni la differenza dei ritardi di accensione = 0B tra il posizionamento degli switch
02CE	85	06			STA	\$06	
02D0	C0	00		LP8	CPY	#S00	
92D2	F0	07			BEQ	ONDLY	
02D4	18				CLC		
02D5	65	06			ADC	\$06	
02D7	88				DEY		Ricicla finché (\$0006) = (lettura switch x \$0B)
02D8	4C	D0	02		JMP	LP8	
02DB	85	06		ONDLY	STA	\$06	
02DD	A9	80			LDA	#S80	Calcolo del ritardo di accensione = 80 + (lettura switch x 0B)
02DF	18				CLC		
02E0	65	06			ADC	\$06	
02E2	85	06			STA	\$06	Memorizza questa costante alla locaz. 0006
02E4	A0	C0			LDY	#SC0	
02E6	A5	06		MTRON	LDA	\$06	Trasferisci (0006) alla loc. 0004 prima di chiamare DLYB
02E8	85	04			STA	\$04	
02EA	A9	00			LDA	#S04	Accendi il motore
02EC	8D	01	AC		STA	\$AC01	
02EF	20	48	02		JSR	DLYB	Quindi chiama DLYB
02F2	A9	C0			LDA	#SC0	Poni la costante del ritardo di spegnimento = C0, indipendentemente dalla lettura degli switch, carica questo alla loc. 0004
02F4	85	04			STA	\$04	
02F6	A9	40		MTROFF	LDA	#S40	Spegni il motore
02F8	8D	01	AC		STA	\$AC01	
02FB	20	48	02		JSR	DLYB	Quindi chiama DLYB
02FE	88				DEY		
02FF	C0	00			CPY	#S00	Ripeti questa sequenza di on-off finché (Y) = 00
0301	30	E3			BMI	MTRON	
0303	4C	C6	02		JMP	MTRSP	Quindi leggi l'impostazione switch e ripeti

Figura 5-43: Controllo del motore (Programma 5-8)

La Figura 5-43 riporta il programma. Le prime quattro istruzioni accendono il motore condizionando il registro di direzione dati e caricando "0" nel registro dei dati:

```
MOTOR    LDA    #$40
          STA    $AC03
          LDA    #$00
AC        STA    $AC01
```

Nella locazione di memoria "00" viene quindi depositato un valore di ritardo "FF", in accordo con la convenzione di passaggio di un parametro alla subroutine DLYA (Vedere programma 5-1). Viene quindi richiamata la subroutine DLYA; essa realizza il ritardo iniziale richiesto affinché il motore acquisti la sua velocità iniziale.

```
          LDA    #$FF
          STA    $00
          JSR    DLYA
```

Viene quindi letto il valore impostato sugli interruttori:

```
          LDA    #$00
          STA    $A003
MTRSP    LDA    $A001
```

Da questa lettura vengono estratti i tre bit di ordine più basso:

```
          AND    #$07      MASCHERA
          TAY
```

Per ogni posizione degli interruttori, eccetto "000", si aggiunge un'unità addizionale di ritardo alla durata minima di "0B" esadecimale. Quindi il valore della lettura degli interruttori viene salvato nel registro indice Y ed il ritardo della durata iniziale viene caricato nella locazione di memoria "06".

```
          LDA    #$0B
          STA    $06
```

LP8 è un ciclo addizionale che aggiunge all'unità di ritardo il numero di volte specificato dal posizionamento degli Interruttori:

```
LP8      CPY    #$00
          BEQ    ONDLY
          CLC
          ADC    $06
          DEY
          JMP    LP8
```

**ESERCIZIO 5-26:** *Siete in grado di modificare il codice precedente in modo che CPY #\$00 non sia indispensabile? Perché?*

Una volta arrivati ad ONDLY, la locazione di memoria "06" contiene la durata addizionale dell'impulso, come specificato dagli interruttori. Viene quindi aggiunta alla durata minima "80" esadecimale:

```
ONDLY    STA    $06
          LDA    #$80
          CLC
          ADC    $06
          STA    $06
```

Il registro Y viene caricato con il valore esadecimale "C0" che specifica il numero di volte che il motore verrà acceso e spento:

```
LDY      #C0
```

Una volta raggiunta la locazione MITRON, la locazione di memoria "06" contiene la costante necessaria a realizzare il ritardo di accensione. Questo viene trasferito nella locazione "04" in modo che possa utilizzarlo la subroutine DLYB. Il motore viene acceso e quindi eseguito il ritardo:

```
MTRON    LDA      $06
          STA      $04
          LDA      #$00          ACCENSIONE DEL MOTORE
          STA      $AC01
          JSR      DLYB
```

Deve quindi essere realizzato il ritardo di spegnimento ed il valore esadecimale "C0" viene caricato nella locazione di memoria "04". Il motore viene spento e realizzato il ritardo per mezzo della subroutine DLYB:

```
MTROFF   LDA      #C0
          STA      $04
          LDA      #$40          MOTORE SPENTO
          STA      $AC01
          JSR      DLYB
```

Dopo che il motore è stato spento viene decrementato il contatore del ciclo Y. In questo caso il registro indice Y viene utilizzato per contare il numero di volte dell'esecuzione del ciclo di accensione/spegnimento. Esso è stato caricato con il valore iniziale esadecimale "C0" esadecimale e viene decrementato ogni volta che il motore viene spento. Quando si raggiunge il valore "0" il programma ritorna a MTRON per eseguire ancora un ciclo di accensione/spegnimento:

```
DEY
CPY      #$00
BMI      MTRON
JMP      MTRSP
```

Consideriamo ora dei miglioramenti del programma.

**ESERCIZIO 5-26:** Eseguiamo inizialmente alcuni miglioramenti nello stile: Si esamini il programma corrispondente agli indirizzi di memoria da 2D0 a 2D8. Sapreste suggerire un miglioramento per la scrittura del codice? (Suggerimento: Si può risparmiare un'istruzione.)

**ESERCIZIO 5-27:** Stessa domanda per le righe da 02FF a 0303.

**ESERCIZIO 5-28:** Questo esercizio è molto interessante se si sta eseguendo un esperimento su un motore reale: se si aumenta progressivamente il ritardo di spegnimento, mediante la variazione della costante appropriata del programma, cosa accade?

**ESERCIZIO 5-29:** Stessa domanda nel caso che si diminuisca il ritardo di accensione. Qual è il problema?

**ESERCIZIO 5-30:** *Si potrebbe realizzare un algoritmo alternativo inviando un numero variabile di impulsi di accensione di durata costante, cioè regolando la durata del ritardo di spegnimento invece di quello di accensione. Siete in grado di modificare conseguentemente il programma?*

**NOTA IMPORTANTE:** Poiché ogni motore ha caratteristiche diverse la miglior temporizzazione del programma può essere determinata con un processo a prova d'errore. Si incoraggia vivamente il lettore a modificare le varie costanti di tempo utilizzate in modo da avere un ritardo di accensione minimo, il minimo ritardo di spegnimento, fornendo incrementi fino ad ottenere l'impostazione caratterizzata dai migliori risultati. Inoltre, collegando il motore ad un carico, cioè ad un dispositivo reale, si introdurranno parametri addizionali di inerzia e frizione. Inoltre, i motori a basso costo per uso hobbistico, possono essere scarsamente lubrificati e, dopo un periodo di poche settimane o pochi mesi, possono presentare una frizione molto più alta. Quindi essi richiederanno un periodo di avviamento molto più lungo ed impulsi più lunghi. Comprendendo il funzionamento meccanico del motore sareste in grado di regolare corrispondentemente i parametri elettrici.

**ESERCIZIO 5-31:** *Che cosa succede se si invia un impulso di accensione molto breve?*

Il programma precedente contiene un ciclo di controllo aperto per la regolazione della velocità del motore ma non esegue la sua misura. Consideriamo i miglioramenti possibili di questa tecnica.

**ESERCIZIO 5-32:** *Si mostri l'impostazione di velocità del motore. L'impostazione di velocità del motore potrebbe essere identica all'impostazione degli interruttori, cioè si potrebbe mostrare un numero compreso tra 0 e 7.*

Come prossimo esercizio si intende realizzare un controllo reale ad anello chiuso. Questo esercizio è particolarmente interessante se si vuole capire il concetto di regolazione di un disk, per esempio. All'albero del disk, contenuto nella sua busta, viene connesso un modo semplice ed efficiente per misurare la velocità del motore. Un foro, chiamato foro indice, è stato pre-forato nel disk. Si dispone il disk in modo che una sorgente luminosa si trovi da un lato del disk stesso e che un ricevitore si trovi dall'altro lato. Essi dovrebbero essere disposti in modo che, quando il foro passa di fronte al diodo emettitore di luce, la luce illumina il ricevitore. (Questa è la disposizione tipica di un floppy disk per la rivelazione del foro indice del disk). Ogni volta che viene illuminato il ricevitore, si formerà un impulso. Contando il numero di impulsi al secondo si ottiene la velocità rotativa esatta del motore in giri al secondo. Impiegando questa informazione è possibile regolare la durata, o la frequenza degli impulsi di accensione o di spegnimento per regolare la velocità con grande precisione. La regolazione di velocità del floppy disk va oltre queste considerazioni in quanto la velocità del floppy disk deve essere regolata con grande precisione anche durante una rotazione parziale del disk. Sul disk si utilizza un'informazione addizionale: l'informazione è registrata sulla traccia e gli impulsi sono impiegati per regolare la velocità di rotazione durante una parte di una rivoluzione singola. Nel caso del motore è importante misurare la velocità effettiva in quanto un carico qualsiasi sul motore modificherà la velocità. Per realizzare questo sono già state introdotte tutte le tecniche hardware e software necessarie.

**ESERCIZIO 5-33:** *Si scriva il programma che realizzi quando detto.*

## **CONVERSIONE ANALOGICO - DIGITALE (UN SENSORE DI CALORE)**

Per la misura della temperatura si utilizzerà un termistore. Si potrebbe utilizzare qualsiasi altro dispositivo in grado di rivelare il calore. La resistenza del termistore varia con la temperatura. Si utilizzerà questa caratteristica per rivelare variazioni di temperatura in un certo ambiente

e per prendere dei provvedimenti in dipendenza della temperatura misurata. Il problema principale è, dato un valore analogico (cioè un valore che varia con continuità, in questo caso la resistenza del termistore), determinare il valore binario corrispondente. Questo problema è denominato conversione analogico-digitale. Attualmente esistono dei dispositivi che eseguono questa conversione praticamente con un solo componente.

In questa sede vogliamo utilizzare la scelta meno dispendiosa (e più istruttiva) che impiega un convertitore digitale-analogico ed alcuni amplificatori operazionali. La conversione analogico-digitale verrà eseguita per mezzo del programma (Per i dettagli sulle tecniche di conversione analogico-digitale si rimanda al Capitolo 5 del libro "Tecniche di Interfacciamento dei microprocessori").

Qui si utilizzerà una *tecnica di approssimazioni successive*. Si genera un valore binario iniziale e lo si converte in forma analogica; successivamente questa approssimazione analogica verrà confrontata, mediante un comparatore, con il valore generato dal termistore. Il risultato del confronto, "0" oppure "1", dipendentemente se è maggiore o minore, verrà impiegato per generare la successiva approssimazione.

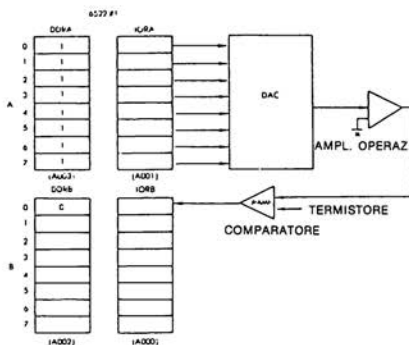


Figura 5-44: Connessione dell'ADC

La Figura 5-44 riporta la connessione hardware utilizzata in questo esperimento: l'uscita ad 8 bit di IOA è connessa ad un DAC ad 8 bit, cioè al convertitore digitale-analogico. Questo convertitore digitale-analogico trasforma il numero binario di 8 bit in un segnale analogico il cui valore viene confrontato con quello del termistore. L'uscita del convertitore ritorna ad IOB, precisamente al bit 0, per essere rivelata.

L'algoritmo considera in successione ogni bit di IOA da quello più significativo (bit 7) fino al bit 0.

Il valore iniziale sarà "10000000". Se esso risulta troppo piccolo si lascia invariato il bit 7 e si pone ad "1" il bit 6. Quindi la successiva approssimazione sarà "11000000". Se a questo punto il valore approssimante è troppo alto (risultato dalla lettura dell'uscita del comparatore) allora si pone nuovamente il bit 6 uguale a "0". L'approssimazione successiva sarà quindi "10100000". Si noti che il bit 5 è stato automaticamente posto ad "1". Quindi il procedimento si ripete.

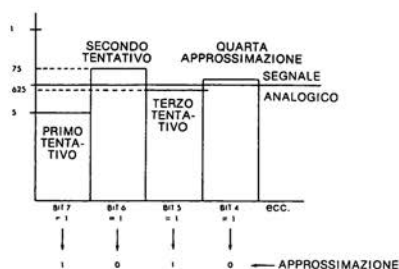


Figura 5-45: Approssimazioni successive

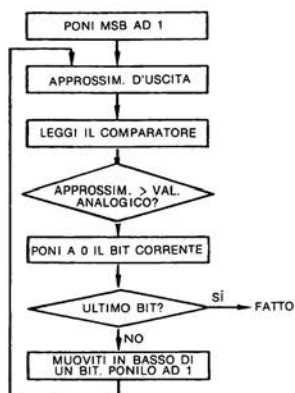


Figura 5-46: Diagramma di flusso delle Approssimazioni Successive

La Figura 5-45 illustra l'algoritmo formale e la Figura 5-46 il diagramma di flusso. Il processo continua fino alla determinazione di tutti gli otto bit. Il valore binario risultante è la migliore approssimazione possibile del valore analogico, con la precisione consentita da una rappresentazione di 8 bit. Naturalmente si assume che l'algoritmo sia sufficientemente veloce in modo che il valore analogico non cambi apprezzabilmente durante la misura. Diversamente occorre utilizzare un circuito *sample-and-hold*. La Figura 5-45 mostra il caso in cui il metodo conduce esattamente al valore analogico. Ogni volta che si impiega un nuovo bit l'intervallo viene diviso per due.



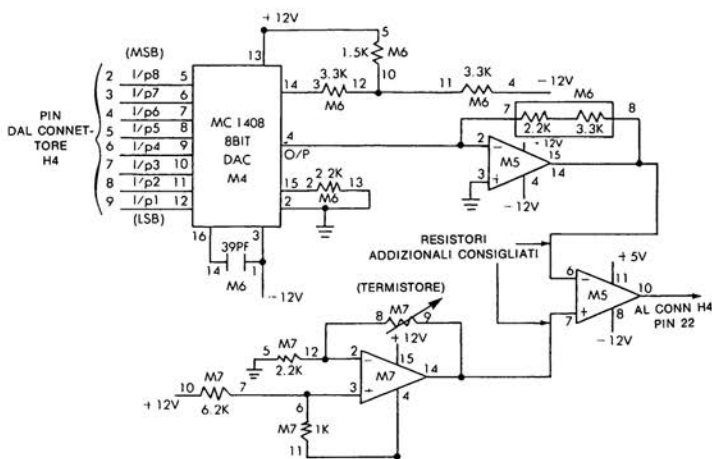


Figura 5-47: Interfaccia dell'ADC

## La Connessione Hardware

Le Figure 5-47 e 5-48 mostrano la connessione hardware. Il DAC impiegato in questo caso è un MC1408, che richiede un'alimentazione di 12 V. La sua uscita pilota l'amplificatore operazionale M5 che alimenta l'ingresso del comparatore. Il termistore appare in basso nell'illustrazione ed alimenta l'altro ingresso del comparatore. L'uscita del comparatore è connessa al pin22 del connettore H4 ed alimenta il bit0 di IORB del 6522#1.

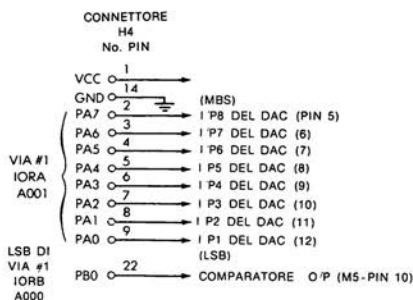


Figura 5-48: Connessione ad H4

## Il Programma

Questo programma è realizzato in modo che il valore della temperatura misurata sul termistore venga rivelato dalla frequenza di un tono su un altoparlante. Il periodo del tono diventerà più alto all'aumentare della temperatura.

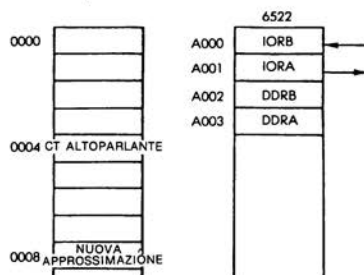


Figura 5-49: Mappa di memoria per l'ADC

La Figura 5-49 riporta la mappa di memoria del programma di conversione analogico-digitale. La locazione di memoria 4 viene utilizzata per immagazzinare la costante impiegata dal programma DLYB, che genera un ritardo in funzione del valore della costante. La locazione 8 viene impiegata per immagazzinare la nuova approssimazione calcolata dal programma. Il 6522 #1 viene rappresentato nelle locazioni di memoria A000 e successive.

La Figura 5-50 rappresenta il diagramma di flusso. Il 6522 viene inizialmente configurato con IORA come uscita del DAC ed il bit 0 di IORB come ingresso del comparatore. Il registro del puntatore viene posto al valore iniziale "10000000" che è il valore approssimante iniziale. Questo registro puntatore punterà al bit da valutare nel ciclo di approssimazione. Ogni volta che è stato completato un ciclo il bit verrà spostato a destra di una posizione.

Il valore di approssimazione iniziale viene posto uguale al registro puntatore. Questo valore viene quindi convertito in analogico. Viene quindi implementato un ritardo per fornire al DAC il tempo sufficiente per la conversione e poi si esamina l'uscita. Se l'uscita del comparatore è "1" allora il valore approssimante è troppo piccolo ed il bit corrente deve rimanere invariato; invece, se l'uscita del comparatore è "0", il valore approssimante è troppo alto ed il bit corrente deve essere posto a "0". Successivamente il registro del puntatore viene fatto scorrere a destra di una posizione di bit in modo da puntare al bit da impiegare nel ciclo successivo. Quando si raggiunge l'ultimo bit si calcola l'approssimazione finale.

Una volta raggiunto il valore dell'approssimazione finale si deve generare un tono con periodo dipendente dal valore della misura. Si utilizza una frequenza di tono minima e la costante del periodo viene ottenuta sommando il valore dell'approssimazione a questa frequenza minima. Per il funzionamento dell'altoparlante viene quindi richiamata la routine dell'altoparlante (BSCSPK). Dopo che l'altoparlante ha suonato per un periodo di tempo minimo il programma legge ancora il valore del termistore.

Sulla scheda il modo più veloce per ottenere un responso udibile è di posizionare il termistore in prossimità della punta del saldatore (o di una sigaretta). Il suono emesso dall'altoparlante dovrebbe aumentare velocemente in periodo. Quando si allontana la sorgente di calore dal termistore l'altoparlante eseguirà una sequenza inversa. Naturalmente il termistore potrebbe essere esterno alla scheda. Opportunamente isolato, esso può essere posizionato su una parete, un contenitore oppure in qualsiasi altro dispositivo del quale si vuole rilevare la temperatura. Per esempio se si vuole controllare la temperatura di un ambiente si può utilizzare un elemento riscaldante connesso ad un relé. Potrebbe rimanere il problema di calibrare il termistore in modo da poter eseguire delle misure precise di temperatura.

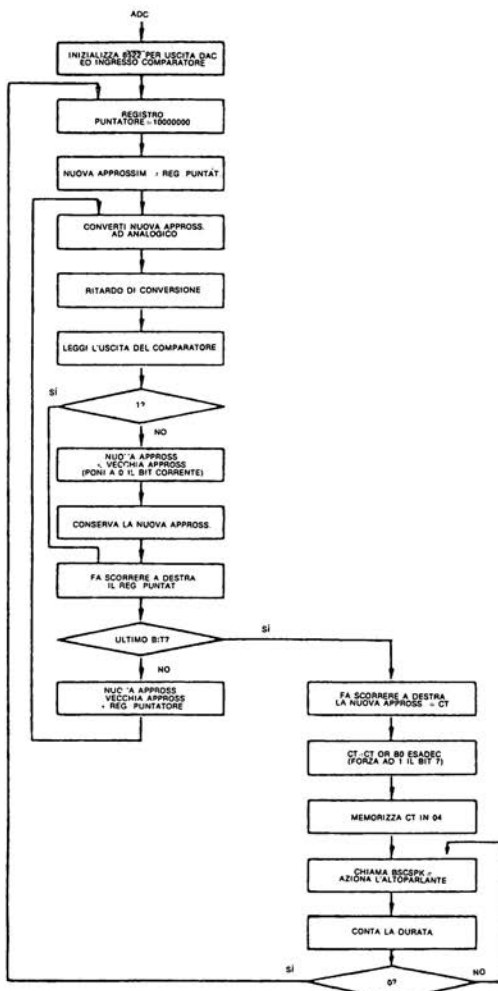


Figura 5-50: Diagramma di flusso per l'ADC

Connessione: Connettore A al connettore H4  
Connettore AA al connettore H3

Questo programma opera per approssimazioni successive con un DAC in modo da rivelare con continuità il valore analogico di un termistore. Quindi il valore digitale approssimato viene impiegato come parametro per controllare la frequenza dell'altoparlante. Dalla variazione di frequenza si può dire se la temperatura sta aumentando o diminuendo.

La frequenza dell'altoparlante è proporzionale alla temperatura (o alla resistenza del termistore)

Questo programma utilizza le subroutine BSCSPK e DLYB.

0360	A9	FF		ADC	LDA	#\$FF	
0362	8D	03	A0		STA	\$A003	Configura VIA #1 DDRA = FF come uscita per pilotare il DAC
0365	A9	00			LDA	#\$00	
0367	8D	02	A0		STA	\$A002	Configura VIA #1 DDRB = 00 come ingresso per leggere il comparatore
036A	A9	80		FSTBIT	LDA	#\$80	Poni MSB per approssimazione
036C	A8				TAY		(Y) memorizza il bit corrente sotto test
036D	85	08			STA	\$08	La loc. 0008 memorizza il valore corrente sotto test
036F	A5	08		NXTBIT	LDA	\$08	
0371	8D	01	A0		STA	\$A001	Uscita del valore corrente al DAC
0374	A2	20			LDX	\$20	Ritardo per posizionamento comparatore
0376	CA			LP9	DEX		
0377	E0	00			CPX	#\$00	
0379	10	FB			BPL	LP9	
037B	AD	00	A0		LDA	#\$A000	Leggi l'uscita del comparatore
037E	29	01			AND	#\$01	Accetta il bit 0
0380	C9	01			CMP	#\$01	
0382	F0	05			BEQ	SHFBIT	Uscita comparatore = 1 significa uscita DAC ancora troppo bassa, conserva il valore corrente e fa scorrere il bit, altrimenti l'uscita del DAC è troppo alta e ricava il bit corrente dal valore corrente, quindi fa scorrere il bit
0384	98				TYA		
0385	45	08			EOR	\$08	
0387	85	08			STA	\$08	
0389	98			SHIFBIT	TYA		
038A	4A				LSR	A	Fa scorrere a destra (Y) di 1 bit per l'approssimazione successiva
038B	A8				TAY		
038C	C9	00			CMP	#\$00	
038E	F0	08			BEQ	ECHO	(Y) = 0 significa approssimazione completata, aziona l'altoparlante
0390	18				CLC		
0391	85	08			ADC	\$08	(Y) = 0, valore corrente più bit successivo come uscita al DAC per l'approssimazione successiva
0393	85	08			STA	\$08	
0395	4C	6F	03	ECHO	JMP	NXTBIT	
0398	A0	F0			LDY	#\$F0	Costante di ritardo per ogni frequenza
039A	A5	08			LDA	\$08	
039C	4A				LSR	A	
039D	85	04			STA	\$04	
039F	A9	80			LDA	#\$80	
03A1	05	04			ORA	\$04	Calcola la costante di frequenza corrispondente e memorizzala alla loc. 0004
03A3	85	04			STA	\$04	
03A5	20	30	02	SPKR	JSR	BSCSPK	Chiama BSCSPK per attivare l'altoparlante
03A8	88				DEY		
03A9	C0	00			CPY	#\$00	
03AB	30	F8			BMI	SPKR	
03AD	4C	6A	03		JMP	FSTBIT	Ripeti per la sequenza di approssimazione successiva

Figura 5-51: Convertitore Analogico-Digitale (Programma 5-9)

Esaminiamo ora il programma per suggerire dei miglioramenti. La Figura 5-51 riporta il programma. Le prime quattro istruzioni condizionano i registri di direzione dati per le porte A e B del 6522 #1, rispettivamente come uscita (con un DAC) e come ingresso (per il comparatore):

```
ADC      LDA      #$FF          DDRA1 = FF = USCITA
          STA      $A003
          LDA      #$00
          STA      $A002        DDRB1 = 00 = INGRESSO
```

Le due istruzioni successive immagazzinano il valore letterale dell'esadecimale "80" nel registro Y. Questo è il registro puntatore che viene impostato al valore iniziale "10000000" in binario.

```
FSTBIT   LDA      #$80
          TAY
```

La locazione di memoria "08" viene quindi riservata per memorizzare l'approssimazione corrente. Essa viene inizializzata ad 10000000:

```
STA      $08
```

Si entra quindi nel ciclo di iterazione principale. L'approssimazione binaria viene prelevata dalla locazione di memoria "08" ed inviata al DAC:

```
NXTBIT   LDA      $08
          STA      $A001
```

Viene quindi realizzato un ritardo per consentire il posizionamento del comparatore:

```
LP9       LDX      #$20
          DEX
          CPX      #$00
          BPL      LP9
```

Viene letta l'uscita del comparatore:

```
LDA      #A000        USCITA COMPARATORE
```

Viene quindi prelevato e controllato il bit 0 di IORB:

```
AND      #$01          BIT 0
CMP      #$01
BEQ      SHFBIT
```

Se esso risulta "1" l'approssimazione è troppo bassa ed il bit successivo deve essere posto ad "1". Se esso è "0" il valore è troppo alto ed il bit corrente deve essere posto a "0":

```
TYA
EOR      $08
STA      $08
```

Dopo aver agglustato, se necessario, il valore dell'approssimazione corrente, si fa scorrere a destra il registro puntatore per il bit successivo dell'iterazione:

```
SHFBIT   TYA
          LSR      A
```

Se si è raggiunto l'ultimo bit si è ottenuta la miglior approssimazione possibile e si salta alla locazione ECHO per azionare l'altoparlante:

```
TYA
CMP    #$00
BEQ    ECHO
```

Altrimenti si pone ad "1" il bit successivo dell'approssimazione si ritorna all'inizio del ciclo:

```
CLC
ADC    $08
STA    $08
JMP    NXTBIT
```

La routine ECHO metterà in azione l'altoparlante in funzione del valore misurato. In questa routine, il registro Y viene impiegato per realizzare il ritardo durante il quale suonerà l'altoparlante. In questo caso esso viene caricato con il valore esadecimale "F0". Il valore dell'approssimazione viene letto dalla locazione di memoria "08" e fatto scorrere a destra di una posizione di bit. Questo significa che il valore dell'ultimo bit dell'approssimazione non avrà influenza sul periodo della nota, impiegando questa tecnica.

Il bit 7 viene forzato al valore "1" in modo che l'altoparlante oscilli ad una frequenza minima garantita per l'udibilità.

Il valore risultante viene memorizzato nella locazione di memoria "04" che viene impiegata per trasferire un parametro alla routine BSCSPK, già presentata:

```
ECHO    LDY    #$F0
        LDA    $08
        LSR    A
        STA    $0A
        LDA    #$80
        ORA    $04
        STA    $04
SPK      JSR    BSCSPK    AZIONA L'ALTOPARLANTE
```

Successivamente viene richiamata la routine ed azionato l'altoparlante alla frequenza specificata. Viene quindi decrementato e controllato il registro Y e l'altoparlante suonerà finché esso non raggiunge il valore "0":

```
DEY
CPY    #$00
BMI    SPKR
JMP    FSTBIT
```

Una volta che l'altoparlante ha suonato per una durata prefissata, il programma ritorna all'inizio dell'approssimazione e rivela ancora lo stato del termistore.

**ESERCIZIO 5-34:** Si mostri in esadecimale il valore dell'approssimazione ottenuta.

**ESERCIZIO 5-35:** È possibile eliminare dal programma l'istruzione "CPY #\$00"?

**ESERCIZIO 5-36:** Calibrate il vostro termistore determinando la misura calcolata corrispondente ad una data temperatura misurata con un termometro. Si memorizzi questo valore in una tabella in modo da poter mostrare il valore della temperatura corrente e non il valore del registro dell'approssimazione.

**ESERCIZIO 5-37:** Si modifichi il programma in modo che l'altoparlante suoni da 1 a 10 volte, in dipendenza della temperatura che sta misurando. A temperatura ambiente esso suonerà una volta. Ad alta temperatura esso suonerà 10 volte. Questo è un modo udibile di comunicare i risultati della misura (con una scarsa precisione).

**ESERCIZIO 5-38:** Dopo aver calibrato il termistore si aggiunga un elemento riscaldante (che può essere acquistato a basso costo presso un magazzino hardware) e si regoli la temperatura di un recipiente d'acqua in modo che rimanga ad un valore costante  $T$ . Attenzione: la maggior parte dei termistori non sono protetti contro l'umidità e quindi occorre collegarli all'esterno del recipiente e non immergerli nell'acqua. Sono comunque disponibili delle termocoppie o dei termistori che sono resistenti all'acqua e possono essere immersi direttamente.

**ESERCIZIO 5-39:** Come ulteriore miglioramento del vostro sistema di allarme domestico (Vedere programma 5-7) si realizzi una routine del ciclo di controllo di base che controlli periodicamente la temperatura. Se la temperatura supera un livello impostato, per esempio  $35^{\circ}\text{C}$ , suonerà un allarme. Questo è un rivelatore d'incendio.

**ESERCIZIO 5-40:** Altra variazione: Si vuole mantenere la punta del saldatore ad una certa distanza dal termistore in modo che quest'ultimo raggiunga una certa temperatura, diciamo  $80^{\circ}\text{C}$ . Modificate il vostro programma in modo che esso accenda velocemente un LED non appena la temperatura è minore del valore impostato e che il LED si accenda lentamente quando ci si avvicina al livello di temperatura desiderato. Si può quindi utilizzare un altro LED per indicare se si è sotto o sopra la temperatura richiesta.

## SOMMARIO

In questo capitolo si sono sviluppate delle applicazioni reali, che vanno da un controllo domestico molto semplice ad un complesso controllo industriale. Alla scheda del microprocessore sono stati connessi diversi dispositivi d'ingresso/uscita, tra i quali gli interruttori, i LED, un motore in c.c., un termistore ed una coppia di foto-ricevitori-emettitori. La scelta dei dispositivi e delle tecniche presentate dovrebbe mettere in grado il lettore di risolvere un grande numero di problemi di controllo reali. Per maggiori informazioni sulle tecniche di interfacciamento specifiche si faccia riferimento al libro "Tecniche di Interfacciamento dei Microprocessori". Inoltre per sviluppare una vera capacità di programmazione si raccomanda vivamente di eseguire gli esperimenti.

Nel prossimo capitolo si realizzeranno le interfacce tra la scheda del 6502 e le periferiche reali del computer.





## CAPITOLO 6

# LE PERIFERICHE

### INTRODUZIONE

Nel corso del capitolo si conatterà la scheda del 6502 a periferiche reali del computer. I programmi di questo paragrafo sono stati ottimizzati per mostrare le tecniche più "elegant" per la risoluzione dei problemi, impiegando le risorse specifiche dei componenti coinvolti.

Inizialmente conatteremo una tastiera a matrice a 16 tasti e realizzeremo un impiego ottimale delle caratteristiche dei registri d'ingresso-uscita per minimizzare il numero di istruzioni richieste per identificare e mostrare il carattere. Successivamente si realizzerà un lettore di nastro di carta (banda) con mezzi di fortuna a basso costo. In questa applicazione la banda di carta può essere semplicemente inserita manualmente attraverso il lettore per essere letto correttamente dal microcomputer. Infine si mostrerà la semplicità di connessione di una microstampante (o di una tastiera ASCII) alla scheda del microcomputer. A questo punto il lettore dovrebbe avere acquisito una certa confidenza con gli artifici richiesti alla risoluzione dei problemi più comuni incontrati nelle applicazioni reali.

Le applicazioni presentate in questa sede sono semplici da realizzare e molto utili. I programmi sono direttamente applicabili al SYM, KIM o AIM65, con variazioni minime.

I programmi sono brevi e consentono un'utile esperienza anche se non si desidera conettere una periferica. Si raccomanda quindi un'attenta lettura di questo capitolo.

### TASTIERA

Si conatterà inizialmente una tastiera esterna a matrice di 16 tasti (detta anche tastiera esadecimale) e si identificherà il tasto che è stato premuto. La Figura 6-1 mostra le connessioni della tastiera: essa è connessa agli 8 bit di IORA del 6522. I bit da 0 a 3 sono connessi alle righe, mentre i bit da 4 a 7 sono connessi alle colonne. Nello schema è stato premuto il tasto dell'intersezione della riga 2 e della colonna 7, connettendo la riga alla colonna.

Il registro di direzione dati è configurato per tutte le uscite. In questo programma si sfrutterà una caratteristica speciale di IORA del 6522. L'IOA è in realtà un registro *bi-direzionale*: si condizioneranno tutte le righe per essere unì e tutte le colonne per essere zeri. Se un tasto viene premuto la riga corrispondente sarà messa a massa dalla colonna attraverso il tasto premuto. Quando si rilegge IORA, il valore "0" nella riga corrispondente verrà scritto nel registro. Nell'esempio che si considera, leggendo IORA dopo aver premuto il tasto, il valore risultante sarà "00001011" in binario oppure "0B" in esadecimale. Impiegando una "tecnica di linea inversa", si scriverà in IORA "1111,0111" in binario oppure "FB" in esadecimale. Poiché la riga numero 2 è "0" (collegata a massa), essa collegherà a massa la colonna 7. Rieseguendo la

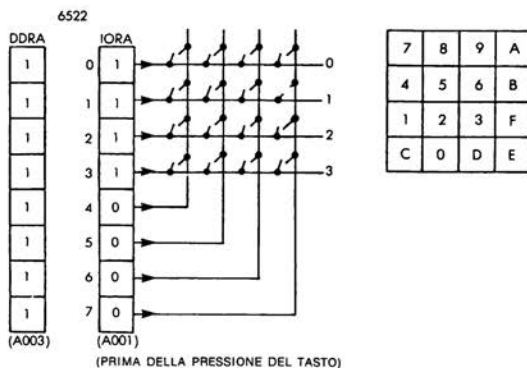


Figura 6-1: Connessione della Tastiera

lettura dei contenuti di IORA, si troverà il valore finale "01111011" in binario oppure "7B" in esadecimale. Ad ogni posizione di bit di IORA dove era presente uno "0", c'è stata la connessione tra la riga o la colonna corrispondente. Questa tecnica non solo rivelerà quale tasto è stato premuto ma anche errori, quale la pressione contemporanea di più tasti. Infatti se viene premuto più di un tasto alla volta, allora sono presenti più zeri per nibble (gruppo di quattro bit) in IORA.

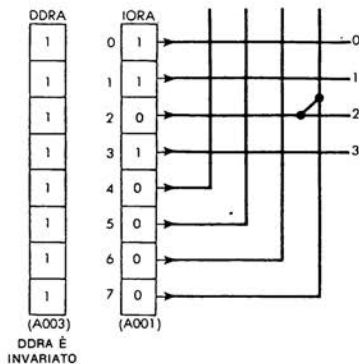


Figura 6-2: Passo 2 — Lettura di IORA dopo la Pressione di un Tasto

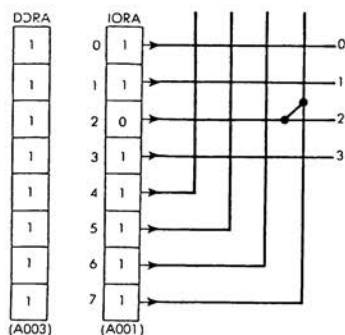


Figura 6-3: Passo 3 – Scrittura di IORA

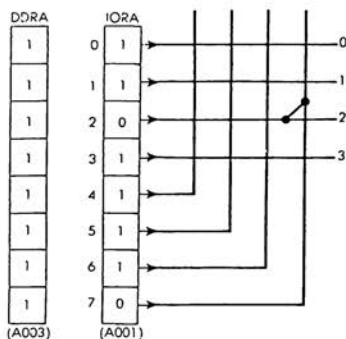


Figura 6-4: Passo 4 – Rilettura di IORA

Per identificare il carattere corrispondente al tasto che è stato premuto (un carattere esadecimale tra "0" ed "F") si costruisce semplicemente una tabella che fornisce la rappresentazione ASCII dei caratteri di ogni pattern consentito in IORA.

Per esempio, abbiamo appena determinato che, quando viene premuto il tasto "B", in IORA si trova il pattern esadecimale "7B". Per esercizio si incoraggia il lettore a calcolare il pattern di IORA per altri caratteri. La Figura 6-5 riporta la tabella corrispondente.

Se si dovesse rivelare un codice non consentito viene ignorato e si procede ad esplorare nuovamente la tastiera.

Infine, una volta ottenuto il codice ASCII del carattere, si procede a mostrarlo sul display.

Per esempio in questo caso, per mostrare il carattere, si utilizzerà l'apposita routine disponibile sul monitor della scheda da SYM. Alla fine del paragrafo si suggeriranno delle modifiche per mostrare il carattere su altri mezzi.

CARATTERE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
HECOD.	DE	ED	DD	BD	EB	DB	BB	E7	D7	B7	77	7B	EE	BE	7E	7D
ASCII	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Figura 6-5: Tabella dei Codici dei Caratteri della Tastiera

NOTA: Questo programma impiega tre routine di monitor: SCAND, HDOUT, ACCESS.

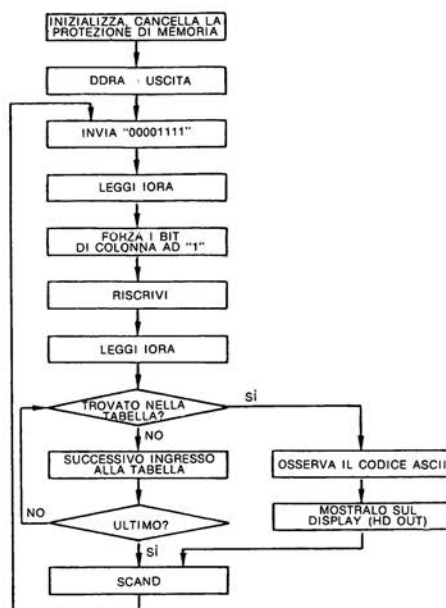


Figura 6-6: Diagramma di flusso della Tastiera

La Figura 6-6 riporta il diagramma di flusso del programma.

Il programma viene inizializzato, quindi si invia su IORA il pattern esadecimale "0F". Il valore di IORA viene riletto (senza cambiare DDRA!). Questo valore non deve essere memorizzato in un registro del 6502 oppure in memoria a causa della caratteristica bidirezionale del compo-

nente di IORA. Esso sarà latched sul componente e rimarrà su di esso. I quattro bit di colonna vengono quindi forzati ad "1" e viene fatto uscire un nuovo pattern di IORA. IORA viene quindi riletto in modo che si possa ottenere il pattern di bit finale. Il pattern nel registro IO viene quindi confrontato ancora con tutti i possibili valori della tabella ASCII della Figura 6-5. Se il codice di IORA non corrisponde all'ingresso corrente della tabella, si passa al successivo. Se nessuno corrisponde si ritorna all'inizio del ciclo.

La Figura 6-7 mostra il programma.

0000	20	86	8B	INIT	JSR	ACCESS	
3	A9	FF			LDA	#\$FF	
5	8D	03	A0		STA	DDRA	DDRA è PAD
8	A2	0F		START	LDX	#\$0F	
A	8E	01	A0		STX	IORA	IORA è PA
D	AD	01	A0		LDA	IORA	IORA è PA
0010	09	F0			ORA	#\$F0	
2	8D	01	A0		STA	IORA	IORA è PA
5	AD	01	A0		LDA	IORA	IORA è PA
8	D5	30		LOOP	CMP	TAB.X	
A	F0	05			BEQ	DISPL	
C	CA				DEX		
D	10	F9			BPL	LOOP	
F	30	05			BMI	SCAN	
0021	B5	40		DISPL	LDA	ASCT.X	
3	20	00	89		JSR	HDOUT	
6	20	06	89	SCAN	JSR	SCAN	
9	4C	08	00		JMP	START	
0030	E7	D7	B7	77 TAB	BYTE	\$E7, \$D7, \$B7, \$77, \$EB, \$DB,	
EB	DB	BB	7B			\$BB, \$7B, \$ED, \$DD, \$BD,	
ED	DD	BD	7D			\$7D, \$EE, \$DE, \$BE, \$7E	
EE	DE	BE	7E				
0040	37	38	39	41 ASCT	BYTE	'7, '8, '9, 'A, '4, '5, '6,	
34	35	36	42			'B, '1, '2, '3, 'F, 'C, '0,	
31	32	33	46			'D, 'E	
43	30	44	45				

Figura 6-7: Programma della Tastiera (Programma 6-1)

La fase di inizializzazione rimuove la protezione della memoria, nel caso della scheda SYM; utilizzando la subroutine di ACCESS, quindi si procede a condizionare il registro di direzione dati della Porta A per predisporre tutte uscite:

```
INIT      JSR      ACCESS
          LDA      #$FF
          STA      DDRA      "11111111" = USCITE
```

Il pattern "00001111" viene quindi inviato al registro dati:

```
START     LDX      #$0F
          STX      IORA      "00001111"
```

Esso viene letto immediatamente e le colonne sono forzate a tutti uni mediante OR con il pattern "11110000":

```
LDA      IORA
ORA      #$F0      "11110000"
```

Il pattern risultante viene inviato al registro dei dati (IORA):

```
STA      IORA
```

Esso viene riletto immediatamente ed ora contiene il pattern finale che verrà impiegato per determinare il tasto che è stato premuto:

LDA IORA

Il codice contenuto nell'accumulatore verrà ora confrontato, in modo sequenziale, con ogni ingresso della tabella. Ogni volta che si ha una struttura a tabella, si utilizza convenientemente il modo di *indirizzamento indicizzato* per accedere in sequenza agli elementi. Il valore iniziale del registro indice è "0F" esadecimale, oppure "15" decimale. Si farà un confronto con l'ultimo ingresso della tabella (Vedere Figura 6-7).

Quindi verrà controllato quello precedente. Ogni volta che il confronto dà esito positivo si verifica un salto alla locazione DISPL:

```

LOOP    CMP    TAB,X
        BEQ    DISPL
        DEX
        BPL    LOOP

```

Se il confronto dà esito negativo si decrementa il registro X e quindi si procede al confronto del carattere successivo. Si deve eseguire il confronto con il valore "0": quando esso diventa negativo non è stato rivelato nessun tasto valido e si verifica un'uscita da SCAN:

BMI SCAN

A questo punto il registro X indica che il carattere è stato riconosciuto. Esso contiene un numero tra "0" e "15". Vogliamo ora convertire questo numero in codice ASCII, richiesto per il display (o la stampante) del carattere riconosciuto:

DISPL LDA ASCT,X

Nella locazione DISPL l'accumulatore viene caricato con il codice ASCII corrispondente al valore del carattere, determinato dal valore del registro indice X. Anche qui viene utilizzata una tecnica di *indirizzamento indicizzato* per questi dati ordinati sequenzialmente (Vedere Figura 6-9). La subroutine HDOUT (del SYM) provvede quindi a mostrare il carattere (routine SCAND per il SYM) prima di riprendere l'esplorazione della tastiera:

```

SCAN    JSR    HDOUT
        JSR    SCAND
        JMP    START

```

Il programma utilizza due tabelle di costanti. La prima è chiamata "TAB": essa contiene la lista dei pattern di bit consentiti che possono apparire su IORA. Il valore del registro indice X, nell'istante di lettura di uno di questi ingressi, determina l'identificazione del tasto che è stato premuto. La seconda tabella utilizzata è denominata "ASCT" e contiene il codice ASCII per ogni digit della tastiera.

Queste due tabelle sono riportate alla fine del programma in Figura 6-7. Si noti che il registro indice X *non* contiene il digit esadecimale reale corrispondente al tasto che è stato premuto. Da una sequenza di confronti con gli elementi di queste due tabelle si ricaverà il codice ASCII corretto per ogni pattern binario consentito trovato nella tabella TAB. Questa è la ragione per cui queste due tabelle nel programma non sono in sequenza esadecimale.

**ESERCIZIO 6-1:** Si costruiscano le due tabelle, TAB ed ASCT di Figura 6-7 in modo che il valore del registro indice X sia sempre uguale al valore esadecimale del tasto che è stato premuto dalla tastiera.

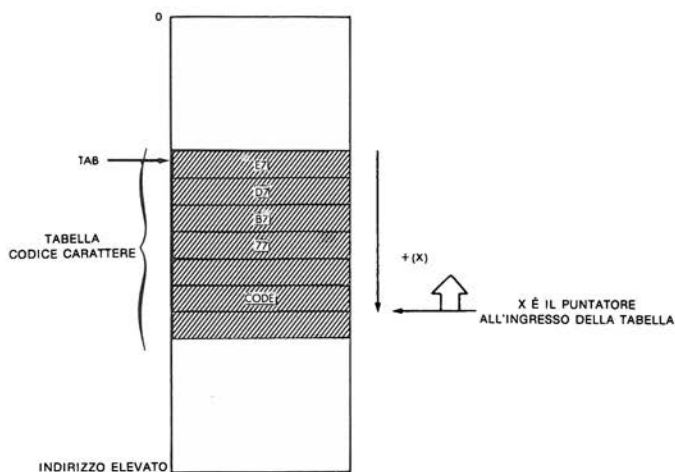


Figura 6-8: Indirizzamento Indicizzato per l'Accesso alla Tabella

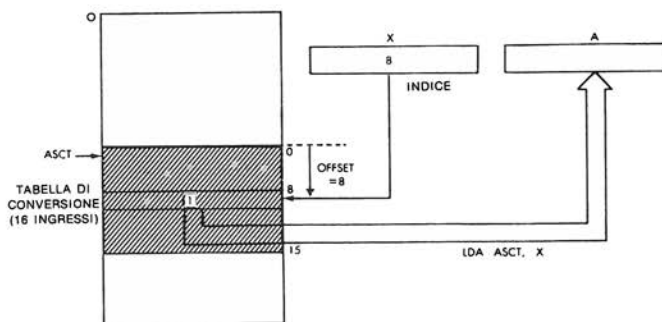


Figura 6-9: Conversione del Carattere ID# al valore ASCII

**ESERCIZIO 6-2:** *Alternativamente al metodo precedente si ridefiniscano i tasti sulla tastiera, senza cambiare le tabelle TAB ed ASCT in modo che il valore del registro indice X corrisponda al tasto che è stato premuto.*

Suggeriamo ora alcune variazioni possibili per mostrare all'esterno, in modo alternativo, il digit che è stato rivelato:

**ESERCIZIO 6-3:** *Si azioni una volta l'altoparlante se è stato premuto il carattere "1", due volte se è stato premuto il carattere "2" e così via.*

**ESERCIZIO 6-4:** *Implegando il programma Morse sviluppato al capitolo 6 (Vedere programma 4-3) si modifichi il programma precedente in modo che esso suoni il codice Morse corrispondente ad ogni tasto premuto.*

**ESERCIZIO 6-5:** *Si modifichi il programma precedente in modo che esso suoni una nota per ogni tasto premuto. Occorre inoltre riservare un tasto per la pausa. Si può utilizzare un altro insieme di due tasti per determinare la durata della nota (durate 1, 2 e 4).*

**ESERCIZIO 6-6:** *Si scriva un programma di musica memorizzata. Inizialmente si suona un accordo premendo i tasti della tastiera nella sequenza desiderata. Le prime 50 note (od un numero qualsiasi) dell'accordo devono essere memorizzate nella memoria del sistema. Quindi, premendo un tasto speciale, il programma deve suonare l'accordo memorizzato.*

## LETTORE DEL NASTRO DI CARTA O TASTIERA ASCII

La connessione di una tastiera decodificata (ASCII) o di un lettore di nastro di carta coinvolge una tecnica pressoché identica. L'interfaccia hardware comprende 8 bit dati (i 7 bit del codice ASCII più il bit di parità) ed un ulteriore bit di stato indicante che il carattere è disponibile. In questa sede verrà presentata una semplice interfaccia per un lettore di nastro di carta semplificato. Il programma per una tastiera decodificata è pressoché identico.

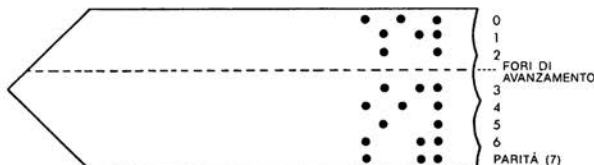
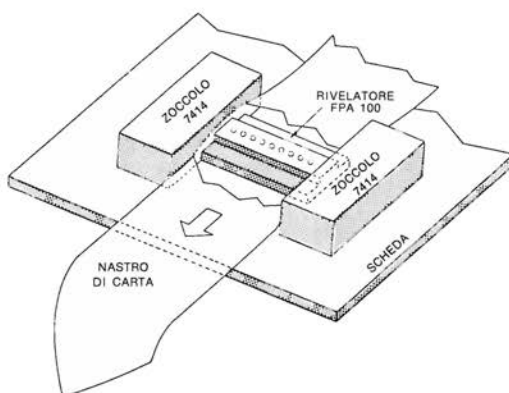


Figura 6-10: Nastro di Carta Perforato ad Otto Livelli

Il nastro di carta è stato tradizionalmente utilizzato per memorizzare programmi in forma affidabile ed economica. Ciascun carattere è rappresentato sul nastro di carta da una fila di fori (Vedere Figura 6-10). Un foro, più piccolo degli altri, viene impiegato per l'avanzamento ed il posizionamento del nastro di carta. Gli altri 8 fori (altri tipi di codice impiegano meno fori) vengono impiegati per la codifica del carattere in formato ASCII. Il nastro di carta viene mosso di una posizione alla volta in modo che il lettore legga il codice in corrispondenza dei fori. In questo caso si utilizzerà una coppia di foto emettitori e rivelatori FPA100.





**Nota:** L'emitter (non indicato) viene posizionato sopra il rivelatore.



*L'emitter FPA 100 viene posizionato alla sommità della piccola scheda. Il PTR è connesso alla scheda del 6502 con un nastro piatto attraverso il connettore A (in alto).*

**Figura 6-11:** Hardware del Lettore del Nastro di Carta

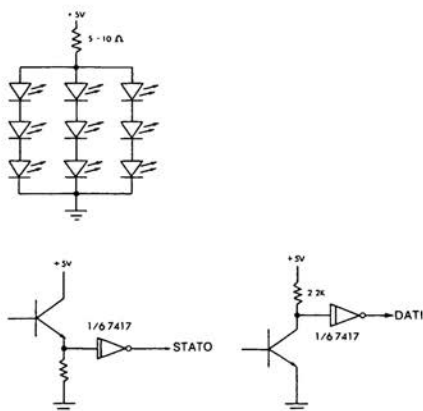


Figura 6-12: Dettagli della Connessione del PTR (Paper Tape Reader: Lettore del nastro di carta)

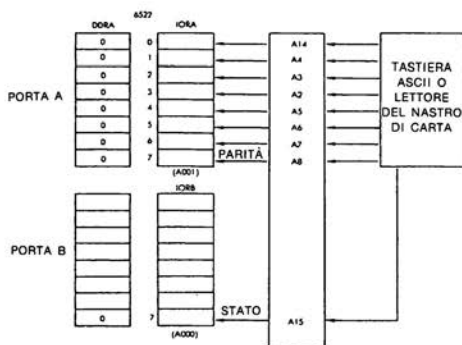


Figura 6-13: Interfaccia del Lettore del Nastro di Carta

I LED emettono luce con continuità: quando un foro viene a trovarsi di fronte al LED si otterrà una trasmissione di luce al fotorelizzatore posizionato dall'altra parte del nastro. Questo sarà un "1". Quando non viene trasmessa nessuna luce si rivelerà uno "0". Si osserva che occorre regolare attentamente l'intensità del LED in modo che non passi luce attraverso il nastro in assenza di fori (in seguito verranno forniti dei consigli pratici). Questo semplice lettore di na-

stro a basso costo può funzionare a mano facendo scorrere il nastro tra i due rivelatori. Il programma sarà auto-sincronizzante. La Figura 6-11 riporta lo schema hardware, mentre la Figura 6-12 riporta i dettagli delle connessioni dei LED e dei rivelatori. La Figura 6-13 rappresenta l'interfaccia del microcomputer. L'IOA del 6522 #1 viene impiegato come ingresso di questi bit dati. L'IOB della Porta B dello stesso 6522 viene impiegato per la lettura del bit di stato nella sua posizione 7.

I segnali sono condizionati mediante Trigger di Schmitt (7414). I due zoccoli del 7414 sono utilizzati come guida del nastro di carta stesso. Il segnale corrispondente alla rivelazione di un foro di avanzamento è "0". Il segnale corrispondente ad un foro dati è "1".

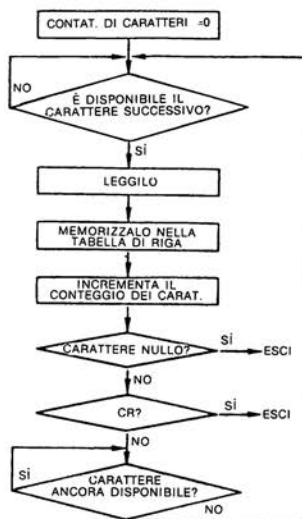


Figura 6-14: Diagramma di flusso del Lettore del Nastro di Carta

Si noti che, in questa semplice interfaccia, viene impiegato un solo resistore per pilotare tutti i LED. In pratica si potrebbero utilizzare resistori singoli per ogni LED. Il valore del resistore deve essere regolato attentamente in modo che il LED fornisca una luce sufficiente per la rivelazione. In caso contrario si rivelerebbero sempre *tutti uni* ("11111111"). Per facilitare la scelta di questo resistore si può considerare inizialmente un nastro di carta *nero*, o almeno molto opaco, in modo da eliminare il problema precedente.

La Figura 6-14 riporta il flow-chart del programma. Per il conteggio dei caratteri entranti si utilizzerà un contatore di caratteri. Il programma rimane in un ciclo di attesa finché non è disponibile il carattere successivo. Questo sarà rivelato dalla presenza di un foro di avanzamento sul rivelatore corrispondente. Una volta che il segnale di stato indica la disponibilità del carattere, occorre procedere velocemente alla sua lettura. Esso viene letto e memorizzato in una tabella in memoria. Successivamente si incrementa il contatore di caratteri.

Per convenzione si assume che l'operazione di lettura termini con un carattere "NULL" (nessun carattere perforato sul nastro), oppure mediante un carattere esplicito di "ritorno carrello" (CR). Quindi il programma cerca il carattere NULL oppure CR e, se lo trova, esce. Se

non lo trova ritorna all'inizio del ciclo. Comunque, prima di ritornare all'inizio del ciclo il programma attende che sia stata ripristinata l'informazione di stato. Una volta scomparso il segnale di "carattere disponibile" si ritorna all'inizio del ciclo e si attende la disponibilità del carattere successivo.

La Figura 6-15 riporta la mappa di memoria corrispondente al programma, mentre la Figura 6-16 riporta il programma.

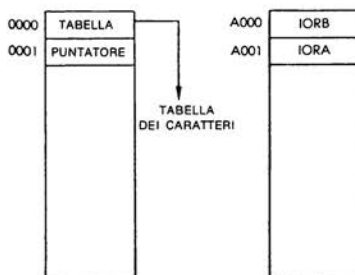


Figura 6-15: Mappa di Memoria per il PTR

```

0002 A0 00 KBPT LDY #0
4 2C 00 A0 TS BIT IORB IORB e PB
7 30 FB BMI TS
9 AD 01 A0 LDA IORA IORA e PA
C 91 00 STA ($00), Y
E C8 INY
F C9 00 CMP #0
0011 F0 0B BEQ RET
3 C9 8D CMP #$8D
5 F0 07 BEQ RET
7 2C 00 A0 TE BIT IORB IORB e PB
A 10 FB BPL TE
C 30 E6 BMI TS
E 60 RET RTS

```

Figura 6-16: Programma PTR/Tastiera (Programma 6-2)

Il programma presuppone che DDRA e DDRB siano stati inizializzati con i valori corretti. In caso contrario occorre aggiungere altre righe di inizializzazione all'inizio di questo programma. Il registro Y viene impiegato come contatore dei caratteri e viene inizializzato al valore "0":

```
KBPT LDY #0
```

Successivamente occorre controllare la riga di stato, per determinare se è disponibile un carattere. Essa è connessa al bit 7 di IORB per facilitare la sua rivelazione:

```
TS BIT IORB
BMI TS
```

Il bit 7 è la posizione ideale per collegare un segnale di stato poiché è una posizione di bit che può essere controllata con una sola istruzione: il bit 7 è quello del "segno". Esso agisce direttamente sul flag "N" del registro di stato, il quale può essere controllato direttamente per "positivo" o "negativo" ("0" oppure "1"). In questo caso esso viene controllato dall'istruzione BMI (salta se meno). Finché questo segnale è "1", nessun carattere è disponibile. Quando esso diventa "0" significa che è disponibile un carattere. L'accumulatore può essere quindi caricato con i dati presenti sulle linee dati:

LDA IORA LETTURA DATI 1

Il carattere di 8 bit ottenuto dal lettore del nastro di carta deve essere quindi memorizzato in una opportuna locazione di memoria. Si assume che l'indirizzo di partenza del buffer di linea sia stato depositato nella locazione di memoria "00, 01". Per accedere al primo elemento della tabella si utilizzerà una tecnica di *indirizzamento indiretto*. Inoltre il modo di indirizzamento sarà *indicizzato* dal valore di Y, per accedere successivamente a tutti gli elementi della tabella. L'istruzione corrispondente è:

STA (\$00),Y

Esaminiamo questa istruzione ad indice indiretto. L'*indirizione* specifica "vai all'indirizzo di memoria "00" ed utilizza i suoi contenuti come indirizzo" (Passo 1 della Figura 6-17).

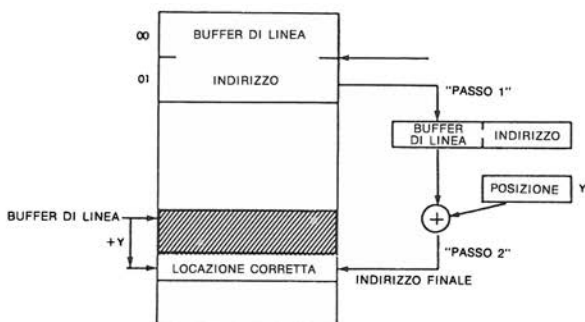


Figura 6-17: Accesso Indiretto Indicizzato: STA (\$00),Y

Quindi il registro Y viene impiegato come indice: i suoi contenuti sono sommati all'indirizzo della base per fornire l'indirizzo finale (Passo 2 della Figura 6-17). I contenuti di Y costituiscono lo spostamento all'interno della tabella del buffer di linea, cioè il puntatore all'ingresso corrente.

Il contatore di caratteri viene quindi incrementato per puntare alla successiva locazione disponibile nel buffer di linea, anticipando il carattere successivo:

INY

Si deve ora controllare il carattere contenuto nell'accumulatore se è un "NULL" oppure un "ri-

torno carrello" per verificare se si è raggiunta la fine di una linea. Questo viene eseguito dalle quattro istruzioni successive:

CMP	#0	NULL?
BEQ	RET	SE SÌ, ESCI
CMP	#\$8D	CR?
BEQ	RET	SE SÌ, ESCI

Infine (con riferimento al diagramma di flusso di Figura 6-14) dobbiamo attendere che il segnale di carattere pronto scompaia per evitare di leggere due volte lo stesso carattere. Questo viene eseguito dalle tre istruzioni successive:

TE	BIT	IOB	CONTROLLO DEL SEGNALE PRONTO
	BPL	TE	
	BMI	TS	

Infine la subroutine termina con la consueta istruzione di ritorno:

RET	RTS
-----	-----

**ESERCIZIO 6-7:** Oltre a memorizzare il carattere in una tabella si generi, attraverso l'altoparlante, il codice Morse corrispondente al carattere letto. Si faccia attenzione a generare il codice Morse in modo sufficientemente veloce da non perdere i caratteri in ingresso. Alternativamente si può decidere di trascinare il nastro molto lentamente in modo da avere il tempo sufficiente per generare l'uscita Morse tra due caratteri successivi. Un'altra soluzione possibile può essere quella di decidere di generare il codice Morse solo alla fine della linea, quando tutti i caratteri sono stati letti. Questa è la soluzione più sicura ma non consente di verificare che ogni carattere sia stato letto correttamente!

**ESERCIZIO 6-8:** Si colleghi otto LED sulla scheda PTR e si accendano con il 6502, per ogni carattere riconosciuto.

**ESERCIZIO 6-9:** Si faccia suonare un allarme se il bit di parità non è corretto. (Il bit di parità assicura che il numero totale di bit per un dato carattere è pari o dispari, in dipendenza della convenzione assunta. Si verifichi questo.)

## MICROSTAMPANTE

Molte piccole microstampanti impiegano una carta elettrosensibile e stampano una linea di 20 caratteri, utilizzando una matrice di punti per formare i caratteri. Gli esempi più comuni sono le Olivetti (diversi modelli) o Matsuhita. Gli elementi stampanti richiedono una piccola interfaccia che invii i segnali corretti alla testina, muova la carta e controlli le parti meccaniche della stampante. Una volta dotata di una tale interfaccia di base, la microstampante può essere connessa a qualsiasi microprocessore dotato di un PIO (una porta d'ingresso/uscita programmabile). Di seguito si collegherà tale stampante al sistema 6502 attraverso un 6522 ed una porta 6532. Si possono riscontrare delle differenze qualora si utilizzi una stampante con un'interfaccia diversa. Comunque la logica del programma rimane essenzialmente la stessa.

Il programma stamperà una fila di 20 caratteri alla volta. Esso sarà avviato dal segnale di inizio stampa ed invierà i 20 caratteri in sequenza. Prima di inviare un carattere il programma attende di ricevere dall'interfaccia della stampante un segnale di richiesta di caratteri. La risposta a questo segnale il programma deve fornire i caratteri, altrimenti il carattere precedente, immagazzinato nel buffer dell'interfaccia, verrà stampato per errore. I caratteri saranno forniti sulle 6 linee dati. Si impiega una rappresentazione dei caratteri a 6 bit (si veda Figura 6-18).

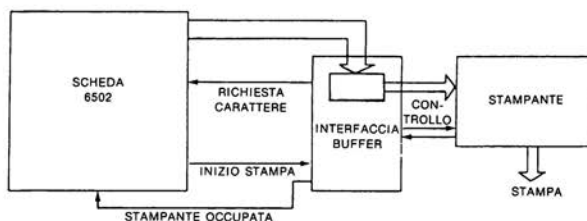


Figura 6-18: Interfaccia di Base della Stampante

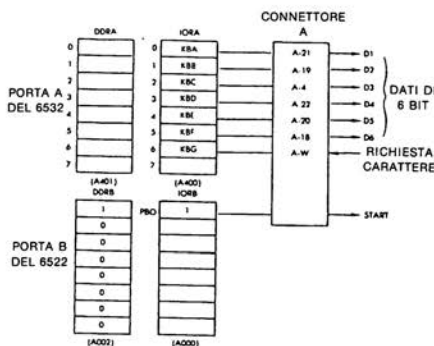


Figura 6-19: Connessione della Stampante

La Figura 6-19 riporta la connessione hardware della stampante. Si utilizzano la Porta A del 6532 ed il bit 0 della Porta B del 6522. L'IOA del 6532 fornisce le 6 linee dati e riceve sul bit 6 la richiesta di carattere, come indicato in figura. Il bit 0 di IOB del 6522 viene utilizzato per generare il segnale di start. Inoltre, normalmente, l'interfaccia della stampante fornisce un segnale di stampante occupata. In questo esempio esso non verrà considerato e si sostituirà con una routine di ritardo software di 30 millisecondi. La Figura 6-20 rappresenta un diagramma di flusso del programma.

I registri di direzione dati dei due PIO vengono inizializzati, viene generato un impulso di start per avviare la stampante. Successivamente il programma controlla la linea di richiesta carattere. Poi il programma attende fino al consenso di richiesta carattere. Esso accetta il carattere successivo dalla locazione di memoria dove è memorizzata la linea di 20 caratteri (Vedere Figura 6-21). Il carattere viene quindi inviato alla stampante, il programma attende che scompaia il segnale di richiesta carattere. Esso incrementa il contatore di caratteri e si verifica se si è raggiunto il valore "20". Se non è stato raggiunto il valore "20" occorre inviare un altro

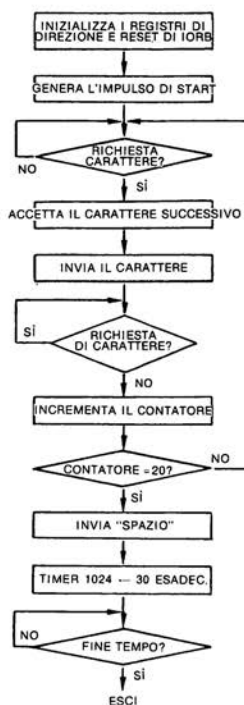


Figura 6-20: Diagramma di flusso del Programma della Stampante

carattere alla stampante e ripercorrere il ciclo. Una volta inviati 20 caratteri alla stampante, si invia un codice di spazio per terminare la linea, originando un avanzamento di una riga ed un ritorno carrello. (Con una diversa interfaccia si può utilizzare una diversa convenzione.) Occorre quindi fornire un ritardo di 48 millisecondi per consentire agli elementi meccanici della stampante di posizionarsi per la riga successiva. Per questo scopo si impiega il timer interno del 6532 ed in questo caso viene utilizzata la parola del timer corrispondente al fattore di tempo 1024. Il fattore 1024 corrisponde ad un ritardo di 1024 microsecondi, ovvero circa 1 millisecondo per l'unità di ritardo di questa parola del timer. Questa parola viene caricata con "30" esadecimale = "48" decimale. Una volta trascorso tale tempo si esce dal programma.

La Figura 6-22 mostra il programma, mentre la Figura 6-21 mostra la mappa di memoria del programma di stampa. Le due locazioni di memoria "00" e "01" contengono il puntatore alla locazione del primo carattere nella memoria. Per impiegare questo programma l'utente dovrebbe caricare il valore "01" alla locazione di memoria "A002" (DDB) e "00" alla locazione



di memoria "A000" (IORB) prima di azionare la stampa. Le locazioni di memoria utilizzate dai dispositivi d'ingresso/uscita sono riportate nella parte destra della Figura 6-19. Consideriamo il programma.

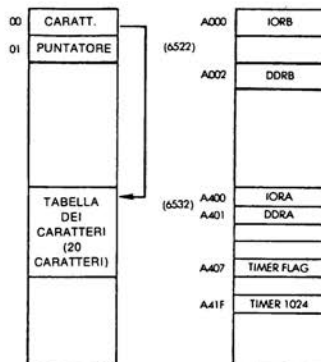


Figura 6-21: Mappa di Memoria per la Stampante

0200	A9	3F		LINE	LDA	#3F	Configura la Porta A
2	8D	01			STA	IORA	
5	A0	01	A4		LDY	#1	Invia il segnale di start
7	8C	00	A0		STY	IORB	
A	88				DEY		
B	8C	00	A0		STY	IORB	Uscita "0"
E	2C	00	A4	TST1	BIT	IORA	Leggi lo stato
0211	70	FB			BVS	TST1	Richiesta carattere?
3	B1	00			LDA	(\$00), Y	Carica carattere
5	8D	00	A4		STA	IORA	Stampalo
8	2C	00	A4	TST2	BIT	IORA	Controlla lo stato
B	50	FB			BVC	TST2	
D	C8				INY		
E	C0	14			CPY	#S14	Il carattere successivo è il 20-esimo?
0220	D0	EC			BNE	TST1	
2	A9	20			LDA	#S20	Spazio carattere
4	8D	00	A4		STA	IORA	
7	A9	30			LDA	#S30	Costante di ritardo
9	8D	1F	A4		STA	T1024	Timer X1024
C	2C	07	A4	TTIM	BIT	TIMFLG	Stato del timer?
F	10	FB			BPL	TTIM	
0231	60				RTS		
0000	50	00			WORD	BUFFER	
0050	30	31 32	33 34	BUFFER	BYTE	'0. '1. '2. '3. '4. '5. '6. '7.	
	35 36	37 38	39 40			'8. '9. 'W. 'A. 'B. 'C. 'D.	
	41. 42	43. 44	45. 46			'E. 'F. 'G. 'H. 'I	
	47 48	49					

IORA è PA  
IORB è PB

Figura 6-22: Programma per la Stampante (Programma 6-3)

Il registro di direzione dati A viene prima inizializzato:

```
LINE      LDA      #$3F
          STA      IORA
```

Depositando il valore "00000001" in IORB viene generato un impulso di start:

```
LDY      #1      "00000001"
STY      IORB
```

IORB viene quindi posto a tutte uscite 0:

```
DEY      Y = "00000000"
STY      IORB
```

Occorre quindi controllare la linea di richiesta carattere. Se questa linea è ad "1" si continua l'osservazione. Quando essa diventa "0" si accetta il carattere successivo:

```
TST1     BIT      IORA      LEGGI LO STATO
          BVS      TST1
```

Come si ricorderà l'istruzione BIT controllerà una certa locazione di memoria senza alterarne i contenuti. Essa copierà rispettivamente il bit 6 e 7 nei flag "V" ed "N". In questo caso si è interessati al valore del bit 6 (con riferimento alla connessione della stampante della Figura 6-19). L'istruzione BVS controllerà il valore del flag di overflow "V", che è stato imposto identico al valore del bit 6 di IORB. Tale valore corrisponde quindi alla linea di richiesta carattere. Il carattere successivo viene ottenuto dalla tabella di 20 caratteri memorizzati negli indirizzi di memoria contenuti nelle locazioni "00" e "01". Un'istruzione di accesso indiretto originerà l'accesso del primo ingresso nella tabella. Per generalità si richiede che questo segmento del programma sia in grado di ricercare un ingresso qualsiasi a questa tabella. Quindi verrà utilizzato, come in qualsiasi organizzazione di tabella, un *indirizzamento indicizzato*. In questo caso, come registro indice, si utilizza il registro Y. Esso contiene inizialmente il valore "00" che verrà incrementato fino al valore 19 prima di uscire dal ciclo. Si utilizza quindi una tecnica di indirizzamento indicizzato:

```
LDA      ($00),Y
```

La Figura 6-23 illustra l'accesso indiretto indicizzato. Inizialmente si accede ai contenuti della locazione di memoria "0001": essi vengono utilizzati come indirizzo della base della tabella. I contenuti del registro Y vengono prima sommati ai contenuti della locazione di memoria 0001 e l'indirizzo finale sarà utilizzato come indirizzo dei dati da prelevare. (Vedere Figura 6-21). Questi dati sono il codice ASCII del carattere da stampare. Esso viene inviato ad IORA:

```
STA      IORA
```

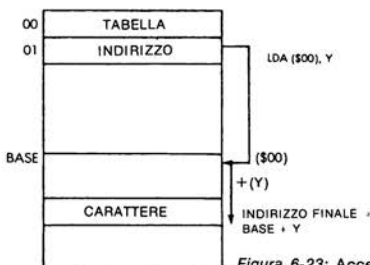


Figura 6-23: Accesso Indiretto Indicizzato

Una volta che il carattere è stato inviato si deve attendere che la linea di richiesta carattere ritorni ancora ad "1". Anche qui si utilizza un ciclo di due istruzioni:

```
TST2      BIT      IORA
          BVC      TST2
```

Viene quindi incrementato il contatore di caratteri (il registro Y):

```
      INY
```

e controllato ancora il valore limite "20" decimale = "14" esadecimale. Finché non si è raggiunto il valore limite si ripercorre il ciclo:

```
      CPY      #$14
      BNE      TST1
```

Su IORA viene quindi fatto uscire il codice richiesto per il carattere "spazio":

```
      LDA      #$20
      STA      IORA
```

Infine si deve garantire il ritardo minimo tra due linee di stampa successive. Per il timer si utilizza il fattore 1024. Il ritardo finale di 48 ms viene ottenuto semplicemente caricando la locazione di memoria opportuna con la costante che specifica il numero di millisecondi (si faccia riferimento alla Figura 6-19 per la mappa di memoria della stampante):

```
      LDA      #$30      DECIMALE = 48
      STA      T1024
```

Quindi viene controllato continuamente il flag del timer finché esso diventa "1", indicando lo scadere del tempo:

```
TTIM      BIT      T1MFLG
          BPL      TTIM
```

La Figura 6-24 riporta la stampa effettiva di una linea di 20 caratteri campione indicata nel programma:

```
0123456789@ABCDEFGHI
```

Figura 6-24: Stampa Effettiva di 20 caratteri

**ESERCIZIO 6-10:** Si connetta la stampante ed il lettore di nastro di carta. La stampante dovrebbe stampare i contenuti del nastro alla fine di ogni linea.

## SOMMARIO

In questo capitolo sono state interfacciate periferiche reali, da un punto di vista hardware e software, alla scheda del microprocessore. È stato fatto un impiego estensivo delle capacità specifiche dei registri del PIO e delle tecniche di indirizzamento consentite dal 6502, in modo da ottimizzare i programmi. Il lettore dovrebbe ora avere appreso tutti gli artifici richiesti per la realizzazione dei propri programmi di applicazione, nei casi più comuni.



## CAPITOLO 7

# CONCLUSIONI

Questo libro ha introdotto sistematicamente le tecniche hardware e software richieste per connettere una scheda reale del 6502 al mondo esterno. Sono stati inizialmente descritti i chip d'ingresso-uscita, assieme alle comuni schede del 6502. Nei capitoli 4, 5 e 6 sono stati successivamente presentati programmi di complessità crescente. A questo punto il lettore dovrebbe avere acquisito confidenza coi dispositivi d'ingresso-uscita che si possono connettere alla scheda del 6502, ed essere in grado di risolvere i problemi di interfacciamento hardware e software. Gli autori ritengono che, sulla base di tutto questo, il lettore dovrebbe essere in grado di risolvere qualsiasi problema applicativo di comune complessità. Ci sono naturalmente dei casi in cui permangono problemi di interfacciamento e si incoraggia il lettore a consultare "Tecniche di Interfacciamento di Microprocessori", per tali problemi. A questo punto, se fossero stati saltati degli esercizi si raccomanda vivamente di ritornare ai capitoli 4, 5 e 6 e di risolvere gli esercizi proposti, prima su carta e poi su una scheda microcomputer reale.

### Il Passo Successivo

Se non si è ancora realizzata nessuna scheda di applicazioni, il passo successivo più logico dovrebbe essere quello di acquistare i pochi componenti a basso costo richiesti per le applicazioni proposte. Si può quindi provare a scrivere da soli i programmi, senza consultare il libro ed assicurarsi di avere acquisito tutto quanto richiesto per risolvere questi problemi. Con un po' di fantasia si possono inventare molte altre applicazioni, impiegando lo stesso hardware limitato oppure qualche altro semplice dispositivo d'ingresso-uscita.

Il lettore interessato a tecniche di programmazione più complesse, richieste per realizzare algoritmi più complessi, viene indirizzato ad un terzo volume della serie, in corso di pubblicazione, "Giochi con il 6502". In questo volume vengono introdotti e descritti degli algoritmi molto più complessi per consentire al lettore di eseguire una grande quantità di giochi che vanno da quelli di abilità ai quadrati magici. L'hardware richiesto per questi giochi è minimo (una tastiera a 16 tasti, 15 LED ed un altoparlante).

È stato constatato che il tempo richiesto per l'apprendimento della programmazione varia notevolmente da una persona ad un'altra. Comunque il passo logico successivo alla lettura di un qualsiasi libro di programmazione dovrebbe essere sempre lo stesso: la pratica. Si spera che i contenuti di questo libro forniscano le basi per la pratica successiva.



## APPENDICE A

# UN ASSEMBLER IN BASIC DEL 6502

### INTRODUZIONE

Lo sviluppo di un breve programma per il 6502 può essere eseguito su carta ed il programma può essere caricato sulla scheda del 6502. Invece se si devono sviluppare dei programmi lunghi (oltre a qualche dozzina di istruzioni), oppure se si deve sviluppare un grande numero di piccoli programmi, conviene un assembler. Poiché si assume che la maggior parte dei lettori siano vivamente interessati ad applicazioni reali del 6502 comincerà a sviluppare tali programmi, si riporta un listing completo di un assembler del 6502 scritto in BASIC per coloro che non dispongono di un assembler del 6502.

Il vantaggio di un assembler del 6502 consiste nel fatto che si può eseguire su qualsiasi computer, dotato di BASIC accessibile dall'utente. La versione di BASIC di questo programma è quello disponibile sul computer Hewlett-Packard. Esso è caratterizzato da un sottoinsieme di BASIC della maggior parte di microcomputer. L'impiego di questo assembler su un computer avente un BASIC diverso coinvolge un processo di traduzione. Comunque lo sforzo di traduzione dovrebbe essere moderato dal fatto che la maggior parte dei BASIC disponibili sui microcomputer comprendono numerose caratteristiche aggiuntive rispetto a quelle impiegate in questo assembler. Per questa ragione questo assembler è praticamente compatibile. Infatti un utente con buona capacità di programmare nel suo BASIC probabilmente sarà in grado di effettuare una riduzione significativa del numero di istruzioni impiegate per questo assembler.

Questo assembler è stato utilizzato per assemblare una grande quantità di programmi per il 6502 e quindi è un prodotto affidabile. Comunque esso viene inserito per soli scopi didattici e non si assumono responsabilità. In un prossimo futuro verrà pubblicata una versione Microsoft BASIC per i lettori interessati a questa particolare versione.

Si riporta un listing completo dell'assembler ed un esempio di funzionamento.

Tutti i programmi riportati alla fine del Capitolo 4 sono stati assemblati con questo assembler.

### DESCRIZIONE GENERALE

L'ASM 65 è un assembler mnemonico completo del 6502. Esso riconosce tutti i mnemonici standard dell'industria, producendo dei listing esadecimali standard, come mostrato nell'esempio di Figura A-1.

Inoltre questo assembler fornisce direttive industriali standard con la sola eccezione dell'impiego del "." per indicare gli assegnamenti e le reference alla locazione corrente. Le direttive disponibili sono: .BYT, .WORD, .DBYT, .TEXT. Per i dettagli di queste direttive si rimanda il lettore alla descrizione dell'assembler di un qualsiasi produttore.

## IMPIEGO DELL'ASSEMBLER

L'ASM 65 è scritto in BASIC Hewlett-Packard 2000 serie F. Di seguito viene riportata una descrizione delle caratteristiche di questa particolare realizzazione BASIC. Per adattare questo interpreter ad altre versioni del BASIC sono necessarie poche variazioni.

L'ASM 65 opera su file seriali. Normalmente sono disponibili almeno tre file e quattro vengono utilizzati. Essi sono: il file sorgente, il file della tabella dei simboli, un file temporaneo ed, opzionalmente, un file di destinazione distinto dal file sorgente.

Il file d'ingresso contiene le istruzioni in linguaggio assembly. Esso deve perciò contenere il testo ASCII e deve essere strutturato in accordo con le regole di sintassi dell'assembler (descritta al paragrafo successivo). In generale le linee d'ingresso possono essere scritte in formato libero, con i campi separati da uno o più spazi. Comunque qualsiasi label deve iniziare in colonna uno. Qualsiasi linea senza label può non iniziare in colonna uno.

```

X CAT SRC
:PROGRAMMA PER IL TRASFERIMENTO DI UN BLOCCO DI MEMORIA
:TRASFERISCE FINO A 255 BYTE DA UNA TABELLA INIZIANTE
:ALLA LOC1 AD UNA TABELLA INIZIANTE A LOC2. LA LUNGHEZZA DELLA
:PARTE TRASFERITA È IN MOVLEN.
MOVLEN =#00
LOC1   =#200
LOC2   =#300
;
      LDX MOVLEN :CARICA NELL'INDICE LA LUNGHEZZA DA TRASFERIRE
LOOP  LDA LOC1,X :CARICA IL BYTE DA TRASFERIRE
      STA LOC2,X :MEMORIZZA IL BYTE DA TRASFERIRE
      DEX :CONTO ALLA ROVERSCIA
      BPL LOOP :SE NON TERMINATO, TRASFERISCI IL BYTE SUCCESS.
      RTS ;DONE

X RUN ASM65
SOURCE FILE ?SRC
OBJECT FILE ?DEST
PRINTOUT ?YES
ASSEMBLY BEGINS...
:PROGRAMMA PER IL TRASFERIMENTO DI UN BLOCCO DI MEMORIA
:TRASFERISCE FINO A 255 BYTE DA UNA TABELLA INIZIANTE A
:LOC1 AD UNA INIZIANTE A LOC2. LA LUNGHEZZA DELLA
:PARTE TRASFERITA È IN MOVLEN.
MOVLEN =#00
LOC1   =#200
LOC2   =#300
;
0000: A6 00      LDX MOVLEN :CARICA NELL'INDICE LA LUNGHEZZA DA TRASFERIRE
0002: BD 00 02  LOOP  LDA LOC1,X :CARICA IL BYTE DA TRASFERIRE
0005: 9D 00 03      STA LOC2,X :MEMORIZZA IL BYTE DA TRASFERIRE
0008: CA          DEX :CONTO ALLA ROVERSCIA
0009: 10 F7       BPL LOOP :SE NON TERMINATO, TRASFERISCI IL BYTE SUCCESS.
000B: 60          RTS ;DONE

SYMBOL TABLE:
MOVLEN      0000      LOC1      0200      LOC2      0300
LOOP        0002
DONE

```

Figura A-1: Impiego dell'Assembler ASM 65



L'assembler predisporrà automaticamente il format del campo del commento sul file d'uscita. Comunque esso non predisporrà il format degli altri campi entro le istruzioni in modo che l'utente può tabulare i suoi statement d'ingresso nel modo più ragionevole per la massima chiarezza. Questa caratteristica serve a migliorare la leggibilità.

Il file d'uscita comprende il testo ASCII con la rappresentazione di tutti i numeri. Il file d'uscita può essere opzionalmente stampato dopo il secondo passo di esecuzione dell'assembler. Sul listing viene stampato un prompt, oppure sullo schermo appare "PRINTOUT?" e l'utente può specificare "si" oppure "no".

L'assembler consente diagnostici estensivi in modo da descrivere tutti gli errori identificati, elencandoli come uscita.

In questa realizzazione la stampa di un errore può contenere vari delimitatori di campi come i limitatori del campo dell'operatore ("!") ed il delimitatore di riferimento interno non definito ("\*\*\*").

La tabella dei simboli fornisce il valore esadecimale di tutte le label simboliche impiegate dal programma. La Figura A-2 riporta un esempio.

SYMBOL TABLE:				
MOVLEN	0000	LOC1	0200	LOC2
LODP	0002			0300
DONE				

Figura A-2: La Tabella dei Simboli

## SINTASSI

### Costanti

Le costanti possono essere rappresentate in una qualsiasi delle quattro rappresentazioni dei numeri:

- Esadecimale: la costante deve essere preceduta da un "\$". Per esempio: "LDA \$20" caricherà l'accumulatore dall'indirizzo di memoria "20" esadecimale.
- Binario: Essa deve essere preceduta da un "%". Esempio: "LDA % 11111111" caricherà l'accumulatore con tutti uni.
- Decimale: rappresentazione normale. Esempio "LDA #0" caricherà l'accumulatore con il valore decimale zero.
- ASCII: deve essere preceduto da un "'". Esempio: "LDA ' A'" caricherà il codice ASCII di A nell'accumulatore.

### Espressioni Aritmetiche

Le espressioni aritmetiche possono essere utilizzate nel campo dell'operatore, per l'assegnazione di una label oppure in un'istruzione di allocazione di memoria.

L'operando di un'espressione aritmetica può essere un numero espresso in una rappresentazione qualunque, una label oppure un "." (il simbolo della locazione corrente) oppure una combinazione di questi. Gli operatori consentiti sono "+" e "-". Nel caso in cui si utilizza più di un operatore, l'espressione aritmetica sarà valutata da sinistra a destra.

### Commenti

I commenti devono essere preceduti da ";". Essi possono iniziare in una colonna qualsiasi, compresa la colonna uno.

## Assegnazioni di Memoria

Le assegnazioni di memoria sono eseguite da una o più delle quattro direttive:

- .BYT — Assegna un byte di dati ad una locazione di memoria.
- .WORD — Assegna due byte di dati a due locazioni di memoria consecutive, basso ordine al primo byte.
- .DBYT — Assegna due byte di dati a due locazioni di memoria consecutive, ordine elevato al primo byte.
- .TEXT — Converte una stringa ASCII in dati esadecimali e li memorizza in locazioni di memoria consecutive. La stringa deve essere delimitata da due caratteri identici non bianchi.

Non esiste la direttiva di fine — al suo posto viene utilizzata quella di fine file.  
Esempio di assegnazione di memoria:

```
.BYT      $2A, WORDCONST
.WORD     2,%10
```

## HP 2000F BASIC

Il BASIC della Hewlett-Packard è diverso da molti BASIC di comuni mini e microcomputer, ma è facilmente adattabile. Di seguito viene riportata una lista di caratteristiche diverse dalla maggior parte dei BASIC o dallo standard di Dartmouth.

### File

I file sono dichiarati in uno statement FILES all'inizio del programma e sono numerati nell'ordine in cui appaiono su di esso. Lo statement ASSIGN assegna un file specifico, mediante il suo primo argomento ad un numero di file specificato dal secondo argomento. Il terzo argomento è una variabile di schermo. Un asterisco che appare in uno statement FILES indica un file che sarà successivamente assegnato a quella posizione da uno statement ASSIGN. Lo statement READ legge il file. Il suo primo argomento, preceduto da un "#" è il numero del file da leggere. Se il numero di record è uno e non c'è punto e virgola lo statement serve per ripristinare il puntatore del file a 0, come in "READ #2,1". Qualsiasi argomento dopo punto e virgola sono le variabili da leggere.

Lo statement PRINT (stampa) è simile a quello di lettura. Esso ha anche una forma speciale, "PRINT #2,END", che contrassegna la fine del file.

Lo statement IF END #THEN opera in modo analogo all'interrupt vettorizzato. Quando si verifica la lettura di fine file l'esecuzione del programma continua al numero di linea indicato dopo THEN, invece di interrompere il programma. Questo si verificherà anche se il computer non sta correntemente eseguendo lo statement; cioè il vettore di fine file deve essere specificato solo una volta, se non deve essere cambiato.

### Stringhe

Le stringhe sono unidimensionali e possono essere dimensionate soltanto come tali. Si utilizza uno statement del tipo "L\$=" per assegnare lunghezza 0 (zero) ad una stringa, cioè per azzerarla. I caratteri di una stringa hanno i seguenti riferimenti: per far riferimento ad una sottostringa all'interno di una stringa più grande si utilizza la forma "T\$(a,b)" dove a e b sono espressioni che indicano rispettivamente gli indirizzi del primo e dell'ultimo carattere nella stringa principale della sottostringa desiderata. I caratteri, all'interno di una stringa, sono indirizzati

da sinistra a destra, partendo da 1. Esempio: se A\$="ABCDE" e viene eseguito lo statement "B\$=A\$(2,3)", allora B\$ diventerà "BC".

La forma "T\$(a)" fa riferimento a tutti i caratteri di T\$ a partire dal carattere #a fino alla fine di T\$.

Esempio: se A\$="12345", A\$(3) indica la sottostringa "345".

Non sono disponibili le funzioni di stringa CHR\$ ed ASC\$ che convertono rispettivamente un numero decimale ASCII in una stringa di un carattere ed una stringa di un carattere nel suo equivalente decimale ASCII, cosicché l'ASM 65 legge una stringa di caratteri ASCII ordinati da un file del sistema chiamato \$ASCII.F, che viene quindi utilizzato per la conversione di numeri e stringhe.

MAX ricava il massimo di due valori.

Esempio: "B=11 MAX 9" produrrà 11.

MIN ricava il minimo di due valori.

LIN se si trova in uno statement di stampa somma un numero di avanzamenti di riga pari al suo argomento.

Le definizioni precedenti servono solo come guida per la traduzione dell'ASM 65 in un'altra versione di BASIC.

Figura A-3: Listing dell'Assembler copyright © 1979, SYBEX Inc.

#### ASM65

```

10 REM : ***** ASSEMBLER MNEMONICO DEL 6502, VERSIONE 2.0 *****
20 REM
30 REM : SCRITTO IN HP2000F TSS BASIC.
40 REM : PUÒ ESSERE UTILIZZATO CON TUTTI I PROCESSORI 65XX PRODOTTI DALLA COMMODORE.
50 REM : SYNERTEK E ROCKWELL
60 REM : TUTTI I MNEMONICI E LE DIRETTIVE SONO STANDARD, CON
70 REM : L'ECCEZIONE DELL'IMPIEGO DI "." PER L'INDIRIZZO CORRENTE
80 R=10
90 T9=0
100 A=0
110 DIM L$(72),M$(72),O$(72),C$(72),Z$(72),P$(72),T$(72)
120 DIM A$(72),N$(72)
130 DIM I$(72)
140 L=0
150 FILES *,SYMTAB,TEMP,*, $ASCII.F
160 PRINT "FILE SORGENTE"
170 INPUT T$
180 PRINT "FILE OGGETTO"
190 INPUT O$
200 ASSIGN T$,1,08
210 ASSIGN O$,4,08
220 READ #1,1
230 PRINT #2,1
240 PRINT #3,1
250 R8=0
260 PRINT "PRINTOUT ";
270 INPUT I$
280 IF I$ <> "NO" THEN 300
290 R8=1
300 PRINT "ASSEMBLY INIZIA..."
310 C=0

```

```

320 IF END #1 THEN 2440
330 L$=""
340 I$=""
350 M$=""
360 O$=""
370 C$=""
380 Z$=""
390 L=L+1
400 ***** SIMBOLI SEPARATI, MEMORIZZA LE ASSEGNAZIONI DI LABEL *****
410 READ #1:I$
420 T5=C
430 IF I$="" THEN 830
440 P=1
450 P$=""
460 GOSUB 3970
470 IF P1=0 THEN 510
480 IF P1=1 THEN 800
490 C$=I$(P1)
500 I$=I$(1,P1-1)
510 IF I$(1,1)="" THEN 590
520 GOSUB 3790
530 L$=P$
540 IF L$ <> "." THEN 590
550 M$="."
560 GOSUB 4940
570 L$=""
580 GOTO 860
590 GOSUB 3790
600 M$=P$
610 IF M$(1,3)=".WO" THEN 3110
620 IF M$(1,3)=".TE" THEN 3110
630 IF M$(1,3)=".BY" THEN 3110
640 IF M$(1,3)=".DB" THEN 3110
650 IF M$ <> "" THEN 850
660 C$=C$(1,34)
670 IF LEN(L$) <> 0 THEN 700
680 I$=I$(1,19)
690 GOTO 820
700 GOSUB 3790
710 N$=P$
720 IF LEN(N$) <> 0 THEN 750
730 T1=C
740 GOTO 780
750 GOSUB 4070
760 IF T4=2 THEN 830
770 T1=F1
780 PRINT #2:L$,T1
790 PRINT #2:END
800 I$=I$(1,LEN(I$) MIN 55)
810 Z$(17,17+LEN(I$))=I$
820 Z$(LEN(I$)+19 MAX 38) MIN 72]=C$
830 PRINT #3:Z$,T5
840 GOTO 320
850 IF M$(1,1) <> "." THEN 1050
860 P$=""
870 GOSUB 3970
880 IF P1>0 THEN 910
890 'OMESSO '-' NELLA LINEA':L
900 GOTO 3090
910 P=P1+1
920 GOSUB 3790
930 IF P$(1,1) <> "" THEN 960
940 'OMESSO ARGOMENTO NELLA LINEA':L
950 GOTO 3090
960 N$=P$
970 GOSUB 4070
980 IF T4 <> 2 THEN 1010
990 'REFERENCE DIRETTA NON CONSENTITA NELLA LINEA' *!L

```

```

1000 GOTO 3090
1010 T1=C
1020 C=F1
1030 IF L$ <> '' THEN 780
1040 GOTO 800
1050 RESTORE 5710
1060 IF M$="" THEN 1140
1070 FOR I=1 TO 56
1080 READ T$
1090 IF T$=M$ THEN 1130
1100 NEXT I
1110 'CODICE OPERATIVO NON NOTO NELLA LINEA' :L
1120 GOTO 3090
1130 O=I
1140 IF L$="" THEN 1170
1150 PRINT #2;L$,C
1160 PRINT #2; END
1170 GOSUB 3750
1180 O$=P$
1190 I$CP-LEN(O$)-1,P-LEN(O$)-1J="I"
1200 "*****"TROVA I MODI DI INDIRIZZAMENTO, CARICA L'INDIRIZZO EFFETTIVO *****
1210 IF O$ <> '' THEN 1240
1220 M=1
1230 GOTO 2200
1240 IF O$ <> 'A' THEN 1270
1250 M=2
1260 GOTO 2200
1270 IF O$[1,1] <> '*' THEN 1320
1280 M=3
1290 P=P+1
1300 N$=O$[2]
1310 GOTO 1870
1320 IF M$[1,1] <> 'B' THEN 1460
1330 IF M$="BIT" THEN 1460
1340 M=12
1350 N$=O$
1360 GOSUB 4070
1370 IF T4 <> 2 THEN 1400
1380 A=-200
1390 GOTO 1970
1400 A=F1-C-2
1410 IF A >= 0 THEN 1430
1420 A=256+A
1430 IF ABS(F1-C) <= 127 THEN 1970
1440 'SALTA OLTRE IL RANGE DELLA LINEA' :L
1450 GOTO 3090
1460 P$="("
1470 P=P-LEN(O$)
1480 GOSUB 3970
1490 P5=P1
1500 P$=","
1510 GOSUB 3970
1520 P6=P1
1530 P7=0
1540 IF NOT P6 THEN 1610
1550 IF I$[P6+1,P6+1] <> 'X' THEN 1580
1560 P7=1
1570 GOTO 1610
1580 IF I$[P6+1,P6+1]="Y" THEN 1610
1590 'MODO DI INDIRIZZAMENTO ERRATO NELLA LINEA' :L
1600 GOTO 3090
1610 IF P5 <> 0 THEN 1780
1620 GOSUB 3790
1630 N$=P$
1640 IF NOT P6 OR NOT P7 THEN 1670
1650 M=5
1660 GOTO 1710

```

```

1470 IF NOT P6 THEN 1700
1480 M=6
1490 GOTO 1710
1700 M=4
1710 GOSUB 4070
1720 A=F1
1730 IF T4 <> 2 THEN 1750
1740 A=-1000
1750 IF ABS(A) <= 255 THEN 1970
1760 M=M+3
1770 GOTO 1970
1780 GOSUB 3790
1790 N$=P$(2)
1800 IF NOT P6 OR NOT P7 THEN 1830
1810 M=10
1820 GOTO 1870
1830 IF NOT P6 THEN 1860
1840 M=11
1850 GOTO 1870
1860 M=13
1870 GOSUB 4070
1880 A=F1
1890 IF (M <> 10 AND M <> 11) OR A <= 255 THEN 1920
1900 'VALORE TROPPO GRANDE PER PAGINA ZERO NELLA LINEA'IL
1910 GOTO 3090
1920 IF T4 <> 2 THEN 1970
1930 A=-1000
1940 IF M=13 THEN 1970
1950 A=-200
1960 ***** STAMPA IL CODICE OPERATIVO & EA SU FILE *****
1970 IF A >= 0 THEN 2070
1980 Z$(10,11)="**"
1990 C=C+1
2000 IF M <> 12 THEN 2020
2010 Z$(11,11)="R"
2020 W9=A+256
2030 IF W9 >= 0 THEN 2200
2040 Z$(13,14)="**"
2050 C=C+1
2060 GOTO 2200
2070 R=16
2080 I=A
2090 GOSUB 4940
2100 T$=A$
2110 A$="000"
2120 A$(4)=T$
2130 IF (M >= 3 AND M <= 6) OR (M >= 10 AND M <= 12) THEN 2180
2140 Z$(13,14)=A$(LEN(A$)-3,LEN(A$)-2)
2150 Z$(10,11)=A$(LEN(A$)-1)
2160 C=C+2
2170 GOTO 2200
2180 Z$(10,11)=A$(LEN(A$)-1)
2190 C=C+1
2200 R=16
2210 I=T$
2220 GOSUB 4940
2230 T$="000"
2240 T$(4)=A$
2250 Z$(1,4)=T$(LEN(T$)-3)
2260 RESTORE 5140
2270 FOR I=1 TO (0-1)*13+M
2280 READ T$
2290 NEXT I
2300 IF T$ <> " " THEN 2370
2310 IF M>6 OR M<4 THEN 2350
2320 M=M+3
2330 C=T$
2340 GOTO 1970

```

```

2350 'MODO DI INDIRIZZAMENTO NON CONSENTITO NELLA LINEA' #L
2360 GOTO 3090
2370 Z%[7,8]=T%
2380 Z%[5,5]=': '
2390 C=C+1
2400 Z%[17,17+LEN(I%)] = I%
2410 Z%[19+LEN(I%)] MAX 38] = C%[1,72-(19+LEN(I%)) MAX 38])
2420 PRINT #3; Z%, TS
2430 GOTO 320
2440 ***** SECONDO PASSO: RISOLVI LE REFERENCE FWD *****
2450 PRINT #2; END
2460 PRINT #3; END
2470 READ #2,1
2480 L=0
2490 READ #3,1
2500 PRINT #4,1
2510 IF END #3 THEN 2870
2520 P=1
2530 READ #3; I%, TS
2540 L=L+1 < 2 THEN 2700
2550 IF I%='' THEN 2850
2560 P%='I '
2570 GOSUB 3970
2580 IF P1=0 OR P1=17 THEN 2610
2590 P=P1
2600 I%[P,P]=' '
2610 IF I%[10,10] <> '*' THEN 2850
2620 GOSUB 3790
2630 N%=P%
2640 IF N%[1,1] <> '(' THEN 2660
2650 N%=N%[2]
2660 GOSUB 4070
2670 IF T4 < 2 THEN 2700
2680 'REF DIR IRRESOLUBILE ' LABEL ERRATA NELLA LINEA' #L
2690 GOTO 3090
2700 I=F1
2710 IF I%[11,11] <> 'R' THEN 2750
2720 I=F1-T5-2
2730 IF I >= 0 THEN 2750
2740 I=I+256
2750 R=16
2760 GOSUB 4940
2770 T% = A%
2780 A% = '000'
2790 A%[4]=T%
2800 IF I%[13,14] <> '***' THEN 2840
2810 I%[13,14]=A%[LEN(A%)-3,LEN(A%)-1]
2820 I%[10,11]=A%[LEN(A%)-1]
2830 GOTO 2850
2840 I%[10,11]=A%[LEN(A%)-1]
2850 PRINT #4; I%
2860 GOTO 2510
2870 PRINT #4; END
2880 IF R8=1 THEN 3080
2890 IF END #4 THEN 2940
2900 READ #4,1
2910 READ #4; I%
2920 PRINT I%
2930 GOTO 2910
2940 READ #2,1
2950 'TABELLA SIMBOLI'
2960 IF END #2 THEN 3080
2970 FOR I6=1 TO 3
2980 READ #2; O%, TS
2990 R=16
3000 I=TS
3010 GOSUB 4940
3020 T% = '0000'

```

```

3030 T$CLEN(T$)+1]=A$
3040 PRINT TAB((I6-1)*25+1);D$;TAB((I6-1)*25+13);T$CLEN(T$)-3];
3050 NEXT I6
3060 PRINT
3070 GOTO 2970
3080 END
3090 PRINT '<'I$>*'
3100 END
3110 ***** PROCESSO DI CARICAMENTO DI MEMORIA *****
3120 Q7=1
3130 IF M$(2,3) <> 'TE' THEN 3260
3140 IF Q7 <> 1 THEN 3190
3150 GOSUB 3750
3160 P=P-LEN(P$)
3170 O$=I$(P,P)
3180 P=P+1
3190 IF P <= 72 THEN 3220
3200 PRINT 'BAD DELIMITER IN LINE '#L
3210 GOTO 3090
3220 P$(1)=' '
3230 P$(2,2)=I$(P,P)
3240 IF P$(2,2)=O$ THEN 320
3250 GOTO 3280
3260 GOSUB 3790
3270 Z$='
3280 P=P+1
3290 IF LEN(P$)=0 THEN 320
3300 N$=P$
3310 GOSUB 4070
3320 IF T4 <> 2 THEN 3350
3330 'LABEL ERRATA NELL'ASSEGNAZIONE DI MEMORIA DELLA LINEA'#L
3340 GOTO 3090
3350 R=16
3360 I=F1
3370 GOSUB 4940
3380 T$=A$
3390 A$='000'
3400 A$(4)=T$
3410 IF M$(2,2) <> 'W' THEN 3460
3420 Z$(10,11)=A$(CLEN(A$)-3,CLEN(A$)-2)
3430 Z$(7,8)=A$(CLEN(A$)-1)
3440 C=C+2
3450 GOTO 3560
3460 IF M$(2,2)='D' THEN 3530
3470 IF F1<256 THEN 3500
3480 'NUMERO TROPPO GRANDE NELL'ASSEGNAZIONE DI MEMORIA DELLA LINEA'#L
3490 GOTO 3090
3500 Z$(7,8)=A$(CLEN(A$)-1)
3510 C=C+1
3520 GOTO 3560
3530 Z$(7,8)=A$(CLEN(A$)-3,CLEN(A$)-2)
3540 Z$(10,11)=A$(CLEN(A$)-1)
3550 C=C+1
3560 I=TS
3570 R=16
3580 GOSUB 4940
3590 T$='000'
3600 T$(4)=A$
3610 Z$(1,4)=T$(CLEN(T$)-3)
3620 Z$(5,5)=' '
3630 IF Q7 <> 1 THEN 3700
3640 IF LEN(L$)=0 THEN 3670
3650 PRINT #2;L$,TS
3660 PRINT #2; END
3670 Z$(17,17+CLEN(I$))=I$
3680 Z$(19+CLEN(I$)) MAX 38]=C$(1,72-(19+CLEN(I$))) MAX 38]
3690 GOTO 3710
3700 Z$=Z$(1,15)

```



```

3710 Q7=0
3720 PRINT #3;Z$,T5
3730 T5=C
3740 GOTO 3130
3750 ***** ROUTINE PER ISOLARE IL SIMBOLO *****
3760 COMINCIA CERCANDO IL SIMBOLO A P. PONILO IN P$ ED
3770 AGGIORNA P. SE L'INGRESSO È TROVATO ARRESTA L'ESPLORAZIONE A ' '.
3780 T9=1
3790 SE L'INGRESSO È QUI ARRESTA LA SCANSIONE A ' ', ' ', ' ', ' '.
3800 FOR I1=P TO LEN(I$)
3810 IF I$(I1,I1) <> ' ' THEN 3830
3820 NEXT I1
3830 P$=' '
3840 FOR I2=I1 TO LEN(I$)
3850 IF I$(I2,I2)=' ' THEN 3920
3860 IF T9=1 THEN 3900
3870 IF I$(I2,I2)='.' THEN 3920
3880 IF I$(I2,I2)='/' THEN 3920
3890 IF I$(I2,I2)='=' THEN 3920
3900 P$[LEN(P$)+1]=I$(I2,I2)
3910 NEXT I2
3920 P=I2
3930 IF LEN(P$) = 0 THEN 3950
3940 P=P+1
3950 T9=0
3960 RETURN
3970 ***** ROUTINE DI RICERCA SIMBOLO *****
3980 RITORNA CON P1 = SYMLOG SE TROVATO, P1 = 0
3990 SE SIMBOLO NON TROVATO
4000 FOR I=P TO LEN(I$)
4010 IF I$(I,I)=P$(1,1) THEN 4050
4020 NEXT I
4030 F1=0
4040 RETURN
4050 P1=I
4060 RETURN
4070 ***** INTERPRETER DI STRINGA NUMERICA *****
4080 SEMPLIFICA LE STRINGHE DI LABEL ED ESPRESSIONI NUMERICHE
4090 DI NUMERI DI BASE QUALSIASI, PIÙ LE COSTANTI ASCII.
4100 F1=W=0
4110 A$=' '
4120 FOR I=1 TO LEN(N$)
4130 IF N$(I,I)='+' THEN 4180
4140 IF N$(I,I)='-' THEN 4180
4150 IF N$(I,I)='*' THEN 4610
4160 A$[LEN(A$)+1]=N$(I,I)
4170 NEXT I
4180 IF A$ <> ' .' THEN 4210
4190 F2=C
4200 GOTO 4480
4210 IF A$[1,1]>'Z' THEN 4350
4220 IF A$[1,1]<'A' THEN 4350
4230 READ #2,T1
4240 IF END #2 THEN 4330
4250 READ #2;T$,T1
4260 IF T$ <> A$ THEN 4240
4270 F2=T1
4280 T4=3
4290 IF END #2 THEN 4320
4300 READ #2;T$,T1
4310 GOTO 4300
4320 GOTO 4480
4330 T4=2
4340 RETURN
4350 IF A$[1,1] <> ' ' THEN 4390
4360 A$=A$[2]
4370 GOSUB 4640
4380 GOTO 4480

```

```

4390 B=10
4400 IF A$(1,1) <> 'Z' THEN 4430
4410 B=2
4420 GOTO 4450
4430 IF A$(1,1) <> '*' THEN 4460
4440 B=16
4450 A$=A$(2)
4460 GOSUB 4750
4470 F2=F
4480 IF W=2 THEN 4510
4490 F1=F1+F2
4500 GOTO 4520
4510 F1=F1-F2
4520 IF I >= LEN(N$) THEN 4610
4530 T$="+-"
4540 FOR W=1 TO LEN(T$)
4550 IF T$(W,W)=N$(I,I) THEN 4590
4560 NEXT W
4570 'OPERATORE ILLEGALE NELLA LINEA';L
4580 GOTO 3090
4590 A$=""
4600 GOTO 4170
4610 T4=0
4620 RETURN
4630 ***** CARATTERE ASCII AL CONVERTITORE DI NUMERO *****
4640 A$=A$(1,1)
4650 F2=0
4660 READ #5,I1
4670 READ #5,I2
4680 FOR I=1 TO 72
4690 IF A$(I,1)=T$(I,I) THEN 4740
4700 F2=F2+1
4710 NEXT I
4720 F2=F2-8
4730 GOTO 4670
4740 RETURN
4750 ***** STRINGA MULTI-RADIX AL CONVERTITORE DI NUMERO *****
4760 B È LA BASE DEL NUMERO IN A$, F È IL PRODOTTO
4770 F=0
4780 I1=0
4790 FOR I2=LEN(A$) TO 1 STEP -1
4800 RESTORE 4910
4810 FOR N=0 TO B-1
4820 READ F$
4830 IF F$=A$(I2,I2) THEN 4870
4840 NEXT N
4850 'NUMERO ERRATO NELLA LINEA';L
4860 GOTO 3090
4870 F=F+N*B^I1
4880 I1=I1+1
4890 NEXT I2
4900 RETURN
4910 DATA '0','1','2','3','4','5','6','7','8','9','A','B','C','D'
4920 DATA 'E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S'
4930 DATA 'T','U','V','W','X','Y','Z'
4940 ***** NUMERO MULTI-RADIX AL CONVERTITORE DI STRINGA *****
4950 I È IL NUMERO D'INGRESSO, R È LA BASE DI CUI A$ SARÀ IL PRODOTTO
4960 A$=""
4970 T=1
4980 FOR N=20 TO 0 STEP -1
4990 IF I/R^N >= 1 THEN 5020
5000 NEXT N
5010 N=N-1
5020 Q=INT(T/R^N)
5030 IF Q <= R-1 THEN 5050
5040 Q=0
5050 T=T-Q*R^N
5060 RESTORE 4910

```

```

5070 FOR S=0 TO Q
5080 READ T$
5090 NEXT S
5100 A$=LEN(A$)+1: T$=T$
5110 IF N=0 THEN 5010
5120 RETURN
5130 ***** TABELLA CODICI OPERATIVI *****
5140 DATA " " " " " " '69' '65' '75' " " '6D' '7D' '79' '61' '71' " " " "
5150 DATA " " " " " " '29' '25' '35' " " '2B' '3D' '39' '21' '31' " " " "
5160 DATA " " '0A' " " '06' '16' " " '0E' '1E' " " " " " " '90' " "
5170 DATA " " " " " " " " " " " " " " " " '80' " "
5180 DATA " " " " " " " " " " " " " " " " 'F0' " "
5190 DATA " " " " " " " " " " " " " " " " " " " " "
5200 DATA " " " " '24' " " '2C' " " " " " " " " '30' " "
5210 DATA " " " " " " " " " " " " " " " " 'D0' " "
5220 DATA " " " " " " " " " " " " " " " " '10' " "
5230 DATA " " " " " " " " " " " " " " " " " " " " "
5240 DATA " " '00' " " " " " " " " " " " " " " " " '50' " "
5250 DATA " " " " " " " " " " " " " " " " '70' " "
5260 DATA " " " " " " " " " " " " " " " " " " " " "
5270 DATA " " '1B' " " " " " " " " " " " " " " " " " " "
5280 DATA " " '0B' " " " " " " " " " " " " " " " " " " "
5290 DATA " " '5B' " " " " " " " " " " " " " " " " " " "
5300 DATA " " '8B' " " " " " " " " " " " " " " " " " " "
5310 DATA " " " " 'C9' 'C5' 'D5' " " 'CD' 'DD' 'D9' 'C1' 'D1' " " " "
5320 DATA " " " " 'E0' 'E4' " " " " " " " " " " " " " "
5330 DATA " " " " 'C0' 'C4' " " " " " " " " " " " " " "
5340 DATA " " " " 'C6' 'D6' " " 'CE' 'DE' " " " " " " " "
5350 DATA " " 'CA' " " " " " " " " " " " " " " " " " "
5360 DATA " " '8B' " " " " " " " " " " " " " " " " " "
5370 DATA " " " " '49' '45' '55' " " '4D' '5D' '59' '41' '51' " " " "
5380 DATA " " " " 'E6' 'F6' " " 'EE' 'FE' " " " " " " " "
5390 DATA " " 'EB' " " " " " " " " " " " " " " " " " "
5400 DATA " " 'CB' " " " " " " " " " " " " " " " " " "
5410 DATA " " " " " " " " " " " " " " " " '4C' " "
5420 DATA " " " " " " " " " " " " " " " " '20' " "
5430 DATA " " " " 'A9' 'A5' 'B5' " " 'AD' 'BD' 'B9' 'A1' 'B1' " " " "
5440 DATA " " " " 'A2' 'A6' " " 'B6' 'AE' " " 'BE' " " " " " "
5450 DATA " " " " 'A0' 'A4' 'B4' " " 'AC' 'BC' " " 'BE' " " " "
5460 DATA " " '4A' " " '46' '56' " " '4E' '5E' " " " " " " " "
5470 DATA " " 'EA' " " " " " " " " " " " " " " " " " "
5480 DATA " " " " '09' '05' '15' " " '0D' '1D' '19' '01' '11' " " " "
5490 DATA " " '4B' " " " " " " " " " " " " " " " " " "
5500 DATA " " '0B' " " " " " " " " " " " " " " " " " "
5510 DATA " " '6B' " " " " " " " " " " " " " " " " " "
5520 DATA " " '2B' " " " " " " " " " " " " " " " " " "
5530 DATA " " " " '2A' " " '26' '36' " " '2E' '3E' " " " " " "
5540 DATA " " " " '6A' " " '66' '76' " " '6E' '7E' " " " " " "
5550 DATA " " '40' " " " " " " " " " " " " " " " " " "
5560 DATA " " '60' " " " " " " " " " " " " " " " " " "
5570 DATA " " " " 'E9' 'E5' 'F5' " " 'ED' 'FD' 'F9' 'E1' 'F1' " " " "
5580 DATA " " '3B' " " " " " " " " " " " " " " " " " "
5590 DATA " " 'F8' " " " " " " " " " " " " " " " " " "
5600 DATA " " '7B' " " " " " " " " " " " " " " " " " "
5610 DATA " " " " '85' '95' " " '8D' '9D' '99' '81' '91' " " " "
5620 DATA " " " " '86' " " '96' " " '8E' " " " " " " " "
5630 DATA " " " " '84' '94' " " '8C' " " " " " " " "
5640 DATA " " 'AA' " " " " " " " " " " " " " " " " " "
5650 DATA " " 'AB' " " " " " " " " " " " " " " " " " "
5660 DATA " " 'BA' " " " " " " " " " " " " " " " " " "
5670 DATA " " '8A' " " " " " " " " " " " " " " " " " "
5680 DATA " " '9A' " " " " " " " " " " " " " " " " " "
5690 DATA " " '9B' " " " " " " " " " " " " " " " " " "
5700 ***** TABELLA MNEMONICI *****
5710 DATA 'ADC' 'AND' 'ASL' 'BCD' 'BCS' 'BEQ' 'BIT' 'BMI' 'BNE'
5720 DATA 'BPL' 'BRK' 'BVC' 'BVS' 'CLC' 'CLD' 'CLI' 'CLV' 'CMP' 'CPX' 'CPY'
5730 DATA 'DEC' 'DEX' 'DEY' 'EOR' 'INC' 'INX' 'INY' 'JMP' 'JSR' 'LDA' 'LDX'
5740 DATA 'LDY' 'LSR' 'NOP' 'ORA' 'PHA' 'PHP' 'PLA' 'PLP' 'ROL' 'ROR' 'RTI'

```

```
5750 DATA *RTS*,"SRC","SEC","SED","SEI","STA","STX","STY","TAX","TAY","TSX"  
5760 DATA *TXA*,"TXS","TYA"  
5770 END
```

## APPENDICE B

### GIOCO DI MOLTIPLICAZIONE: IL PROGRAMMA

```

;***** MULT *****
;
N      = $00
P      = $01
NSAVE  = $02
PSAVE  = $03
T       = $04
D       = $9B
X       = $200
Y       = $201
RESUL   = $202
ASAVE   = $240
XSAVE   = $241
YSAVE   = $242
PA       = $1700
PAD      = $1701
TIMER   = $1707
;
; = $20
0020: A5 00      START LDA N
0022: B5 02      STA NSAVE
0024: A5 01      LDA P
0026: B5 03      STA PSAVE
0028: A9 01      LDA #$01
002A: B0 01 17   STA PAD
002C: 20 50 02   M1 JSR SOUND
0030: 20 90 00   JSR DL250
0032: C6 00      DEC N
0034: D0 F6      BNE M1
0036: A2 14      LDX #14
0038: 20 9F 00   JSR TIME10
003A: 20 50 02   M2 JSR SOUND
003C: 20 90 00   JSR DL250
003E: C6 01      DEC P
0040: D0 F6      BNE M2
0042: A9 00      AGAIN LDA #0
0044: B5 04      STA T
0046: AD 00 17   FUL LDA PA
0048: 30 FF      BMI FOL
004A: E6 04      PLUS1 INC T
004C: A0 00 17   M3 LDA PA
004E: 10 FF      BPL M3
0050: A0 1F      LDY #1F
0052: A2 01      LDX #1
0054: 20 9F 00   M4 JSR TIME10
0056: AD 00 17   LDA PA
0058: 10 FF      BPL PLUS1
005A: 8B        DEY
005C: 8B

```

```

0063: 10 F3          BPL M4
                :RISPOSTA COMPLETA: RISULTATO IN T
0065: A4 02          LDX NSAVE
0067: A4 03          LDY PSAVE
0069: 20 10 02      JSR MULTI          :RISULTATO IN A
006C: C5 04          CMP T
006E: F0 0D          BEQ BRAVO
                :RISPOSTA ERRATA
0070: A0 10          LDY #10
0072: 20 50 02      M5 JSR SOUND
0075: 20 90 00      JSR DL250
0078: 88            DEY
0079: D0 F7          BNE M5
007B: F0 C9          BEQ AGAIN
                :RISPOSTA CORRETTA
007D: A0 20          BRAVO LDY #20
007F: 20 50 02      M6 JSR SOUND
0082: 80            DEY
0083: D0 FA          BNE M6
0085: 00            BRK
                ;
                ;.=590
0090: 98            DL250 TYA
0091: A2 3D          LDX #3D
0093: A0 00          DL 2 LDY #0
0095: C8            DL 1 INY
0096: D0 FD          BNE DL1
009B: E8            INX
0099: D0 FB          BNE DL2
009B: AB            TAY
009C: 60            RTS
                ;
                ;.=59E
009E: 86 9D          TIME10 STX D          :DURATA IN DECIMI DI SECONDO
00A0: A9 62          T0 LDA #62          :98 BASE DIECI
00A2: 80 07 17      STA TIMER          :TIMER 1024
00A5: AD 07 17      T1 LDA TIMER
00A8: 10 FB          BPL T1
00AA: C6 9D          DEC D
00AC: D0 F2          BNE T0
00AE: 60            RTS
                ;
                ;.=5210
0210: BE 00 02      MULTI STX X
0213: 8C 01 02      STY Y
0216: A0 08          LDY #8
0218: A9 00          LDA #0
021A: 4E 00 02      ONE LSR X
021D: 90 04          ONE BCC TWO
021F: 18            CLC
0220: 60 01 02      TWO ADC Y
0223: 4A            TWO LSR A
0224: 6E 02 02      TWO ROR RESUL
0227: 88            DEY
0228: D0 F0          BNE ONE
022A: AD 02 02      TWO LDA RESUL
022D: 60            RTS
                ;
                ;.=5250
0250: 80 40 02      SOUND STA ASAVE
0253: BE 41 02      STX XSAVE
0256: 8C 42 02      STY YSAVE
0259: A9 00          LDA #0
025B: A2 80          LDX #80
025D: A0 00          CL 2 LDY #0
025F: C8            CL 1 INY
0260: D0 FD          BNE CL1
0262: 49 01          EOR #1

```

```

0264: 8D 00 17      STA PA
0267: E8            INX
0268: D0 F3        BNE CL2
026A: A0 40 02      LDA ASAVE
026D: AE 41 02      LDX XSAVE
0270: AC 42 02      LDY YSAVE
0273: 60            RTS

```

# SYMBOL TABLE:

N	0000	P	0001	NSAVE	0002
PSAVE	0003	T	0004	U	0005
X	0200	Y	0201	RESUL	0202
ASAVE	0240	XSAVE	0241	YSAVE	0242
PA	1700	PAD	1701	TIMER	1707
START	0020	M1	002D	M2	003C
AGAIN	0046	POL	004A	PLUS1	004F
M3	0051	M4	0058	M5	0072
BRAVD	007D	M6	007F	DL250	0090
DL2	0093	DL1	0095	TIME10	009E
T0	00A0	T1	00A5	MULTI	0210
ONE	021A	TWO	0223	SOUND	0250
CL2	025D	CL1	025F		
ONE					

## APPENDICE C

### LISTING DEI PROGRAMMI (Capitolo 4, Parte 1)

```

LINE #LOC CODE LINE
0002 0000 ;QUESTA È UNA SUBROUTINE CHE ACCETTA I CARATT. ASCII
0003 0000 ;DA 2CH A 5AH (PIÙ 20H PER SPAZIO) E SUONA
0004 0000 ;IL COD. MORSE EQUIV. SU UN ALTOPARLANTE COLLEGATO A
0005 0000 ;PB7, 6522-025. INOLTRE ESSO COMMUTA ON OFF PB0, 6522-
0006 0000 ;025 E, CON UN DRIVER, QUESTO BIT PUÒ AZIONARE UN
0007 0000 ;TRASMETTITORE. UN PROG. PRINC. CHIAMERÀ QUESTA SUB
0008 0000 ;CON IL CARATT. ASCII IN ACC.
0009 0000 ;ESEMPLI SI POSSONO AVERE
0010 0000 ;ACCETTANDO INGRESSI DA UN TERMINALE PER AVERE IL CODICE MORSE
0011 0000 ;CON QUESTO PROG. OPPURE GENERARE UN ORDINE CASUALE
0012 0000 ;DI CARATT. PER UN CODICE
0013 0000 ;IL FORMATO DEI CARATT. DEL COD. MORSE DELLA TABELLA
0014 0000 ;È: DA SINISTRA A DESTRA, IL PRIMO BIT
0015 0000 ;UGUALE AD 1 È IL BIT DI START ED OGNI
0016 0000 ;1 SEG. È UNA LINEA ED OGNI 0 UN PUNTO.
0017 0000 SPEED = $F0
0018 0000 COUNT = $F1
0019 0000 CHAR = $F2
0020 0000 ; = $300
0021 0300: C9 20 MORSE CMP # $20 ;SE UNO SPAZIO, ESEGUI LA ROUTINE DI SPAZIO
0022 0302: F0 67 BEQ SPACE
0023 0304: C9 2C CMP # $2C ;VEDI SE IL CODICE ASCII
0024 0306: 90 4E BCC EXIT ; È MINORE DI 2CH, E RITORNO SE COSÌ
0025 0308: C9 5B CMP # $5B ;VEDI SE IL CODICE ASCII È MAGGIORE
0026 030A: B0 4A BCS EXIT ; DI 5AH E RITORNO SE COSÌ
0027 030C: AA TAX ;PONI IL CODICE NEL REGISTRO INDICE
0028 030D: B0 45 03 LDA TABLE - $2C,X ;ACCETTA IL CARATTERE MORSE
0029 0310: A0 08 LDY # $8 ;NUMERO DI BIT DA RUOTARE L'ACCUMULATORE
0030 0312: 84 F1 STY COUNT
0031 0314: 0A STARTB ASL A
0032 0315: C6 F1 DEC COUNT
0033 0317: 90 FB BCC STARTB ;FA SCORRERE FINCHÉ NON TROVI IL BIT DI START
0034 0319: 85 F2 STA CHAR
0035 031B: A5 F2 NEXT LDA CHAR
0036 031D: 0A ASL A ;ORA EMETTE IL CODICE MORSE (1 - LINEA, 0 - PUNTO)
0037 031E: 85 F2 STA CHAR
0038 0320: A0 01 LDY # $1 ;PUNTO = 1 PERIODO DI TEMPO. SALTA AL PUNTO
0039 0322: 90 02 BCC SEND ;SE CARRY È 0, PUNTO
0040 0324: A0 03 LDY # $3 ;ALTRIMENTI LINEA (3 PERIODI DI TEMPO)
0041 ;QUESTA PARTE INVIA UN'USCITA ALTA PER N DI PERIODI DI TEMPO (REG. Y) ED
0042 ; UN'USCITA BASSA PER 1 PERIODO
0043 0326: A9 C0 SEND LDA # $C0
0044 0328: 8D 0B A0 STA $A00B ;PONI IL MODO DEL TIMER A FREE RUNNING PER

```

Programma 4-1: Morse (Figura 4-31 del testo)



```

0045 032B: A9 00      LDA # $0          ; QUESTO VALORE
0046 032D: 8D 06 A0    STA $A006
0047 0330: A9 04      LDA # $04          ;E QUESTO VALORE DETERMINA IL TONO
0048 0332: 8D 07 A0    STA $A007        ; DELL'USCITA (CIRCA 1000 MHz)
0049 0335: 8D 05 A0    STA $A005        ;QUESTO FA PARTIRE IL TONO
0050 0338: A9 01      LDA # $1          ;PONI AD 1 IL BIT D'USCITA PB0
0051 033A: 8D 00 A0    STA $A000
0052 033D: 20 57 03    JSR DELAY        ;RITARDO PER UN PERIODO DI TEMPO
0053 0340: A9 00      LDA # $0
0054 0342: 8D 0B A0    STA $A00B        ;PONI A 0 TONF
0055 0345: 8D 00 A0    STA $A000        ;PONE A 0 IL BIT D'USCITA (PB0)
0056 0348: A0 01      LDY # $01
0057 034A: 20 57 03    JSR DELAY        ;RITARDO PER 1 PERIODO DI TEMPO (SPAZIO TRA ELEMENTI)
0058 034D: C6 F1      DEC COUNT        ;DECREMENTA CONTEGGIO - VEDI SE SONO RUOTATI 8 BIT
0059 034F: D0 CA      BNE NEXT          ;SE NO ESEGUI UN ALTRO ELEMENTO
0060 0351: A0 02      FINISH LDY # $2    ;RITARDO PER 3 PERIODI TEMPO (2 QUI PIÙ LO SPAZIO
0061 0353: 20 57 03    JSR DELAY        ; ALLA FINE DELL'ULTIMO ELEMENTO)
0062 0356: 60          EXIT RTS
0063          ; RITARDO PER (REG. Y) * SPEED * 0.005 SEC
0064 0357: 98          DELAY TYA
0065 0358: 0A          ASL A
0066 0359: 0A          ASL A
0067 035A: A8          TAY
0068 035B: A5 F0 D3    LDA SPEED
0069 035D: A2 FA D2    LDX # $FA
0070 035F: CA D1      DEX
0071 0360: D0 FD      BNE D1
0072 0362: 38          SEC
0073 0363: E9 01      SBC # $1
0074 0365: D0 F6      BNE D2          ;RITARDO PER 7 PERIODI DI TEMPO
0075 0367: 88          DEY          ; (SPAZIO TRA PAROLE)
0076 0368: D0 F1      BNE D3          ;RITORNO DAL PROGRAMMA MORSE
0077 036A: 60          RTS
0077 036B: A0 07 SPACE LDY # $7
0077 036D: 20 57 03    JSR DELAY
0077 0370: 60          RTS
0077 0371: 73          TABLE .BYTE $73, $31, $6A, $32, $3F, $2F
0077 0372: 31
0078 0373: 6A
0078 0374: 32
0079 0375: 3F
0078 0376: 2F
0078 0377: 27          .BYTE $27, $23, $21, $20, $30, $38
0078 0378: 23
0079 0379: 21
0079 037A: 20
0079 037B: 30
0079 037C: 38
0079 037D: 3C          .BYTE $3C, $3E, $01, $01, $01, $01
0079 037E: 3E
0080 037F: 01
0080 0380: 01
0080 0381: 01
0080 0382: 01          .BYTE $01, $4C, $01, $05, $18, $1A
0080 0383: 01
0080 0384: 4C
0081 0385: 01

```

Programma 4-1: Morse (continua)

```

0081 0386: 05
0081 0387: 18
0081 0388: 1A
0081 0389: 0C      .BYTE $0C, $02, $12, $0E, $10, $04
0081 038A: 02
0082 038B: 12
0082 038C: 0E
0082 038D: 10
0082 038E: 04
0082 038F: 17      .BYTE $17, $0D, $14, $07, $06, $0F
0082 0390: 0D
0083 0391: 14
0083 0392: 07
0083 0393: 06
0083 0394: 0F
0083 0395: 16      .BYTE $16, $1D, $0A, $08, $03, $09
0083 0396: 1D
0084 0397: 0A
0084 0398: 08
0084 0399: 03
0084 039A: 09
0084 039B: 11      .BYTE $11, $0B, $19, $1B, $1C
0085 039C: 0B
0085 039D: 19
0085 039E: 1B
0085 039F: 1C

```

SYMBOL TABLE:

SPEED	00F0	COUNT	00F1	CHAR	00F2
MORSE	0300	STARTR	0314	NEXT	031B
SEND	0326	FINISH	0351	EXIT	0356
DELAY	0357	D3	035B	D2	035D
D1	035F	SPACE	036B	TABLE	0371

*Programma 4-1: Morse (continua)*

LINE	#LOC	CODE	LINE
			:CARICA PRELIM. A7 NELLA LOC. A67E E 03 IN A07F
0002	0000		:QUESTA È UNA ROUTINE DI OROLOGIO IN TEMPO REALE
0003	0000		:CHE CONSERVA IL TEMPO CORRENTE ALLE LOC. SEC (00F6), MIN
0004	0000		:(005F) ED HOUR (00F4). ESSA È INTERROTTA
0005	0000		:DA FINE TEMPO O INTERRUPT TIMER, CHE
0006	0000		:CAUSA UN INTERRUPT E SALTA ALLA ROUTINE
0007	0000		:DELL'OROLOGIO 20 VOLTE AL SECONDO. LA ROUTINE DELL'OROLOGIO
0008	0000		:ED IL TIMER DELL'INTERVALLO DEVONO PRIMA ESSERE INIZ.
0009	0000		:QUESTO VIENE ESEGUITO DAL COD. 'INIT' E SI DEVE SALTARE
0010	0000		:ALL'INIZIO PER INIZIALIZZ., INTRODURRE IL TEMPO ATTUALE
0011	0000		:LA ROUTINE PARTIRÀ A SEC, MIN ED
			:HOUR INVIANDO IL COMANDO 'GO 0390 CR'
			:NON OCCORRE FARE ALTRO
0012	0000	COUNT = \$00F7	:CONTATORE PER VENTESIMI DI SECONDO
0013	0000	SECS = \$00F6	:TEMPO CORRENTE
0014	0000	MIN = \$00F5	
0015	0000	HOUR = \$00F4	
0016	0000	ACR = \$A00B	:REGISTRO TIMER MODE
0017	0000	T1LL = \$A006	:COSTANTE DEL TIMER DI BASSO ORDINE
0018	0000	T1HC = \$A005	:COSTANTE DEL TIMER DI ORDINE ELEVATO
0019	0000	* = \$0390	
0020	0390	A9 14 INIT LDA #\$14	:POSIZIONATI AI PRIMI VENTI
0021	0392	85 F7 STA COUNT	:CONTEGGI
0022	0394	8D 0B A0 STA ACR	:PONI A 0 IN ACR 1
			:BIT 7 ED 8.
0023	0397	A9 C0 LDA #\$C0	:PONI AD 1 I BIT 8 E 7
0024	0399	8D 0E A0 STA \$A00E	:NEL REGISTRO DI ABILITAZIONE
			:INTERRUPT (PER ABILITARE
			:GLI INTERRUPT DAL TIMER 1)
0025	039C	A9 50 LDA #\$50	:MEMORIZZA C350 NEL TIMER
0026	039E	8D 06 A0 STA T1LL	: (COSTANTE DI RITARDO
0027	03A1	A9 C3 LDA #\$C3	: PER 50 MS)
0028	03A3	8D 05 A0 STA T1HC	:QUESTO AVVIA IL TIMER
0029	03A6	60 RTS	:RITORNA AL MONITOR
0030	03A7	08 CLOCK PHP	:SALVA LO STATO
0031	03A8	48 PHA	
0032	03A9	F8 SED	
0033	03AA	A9 50 LDA #\$50	:MEMORIZZA C350 NEL TIMER
0034	03AC	8D 06 A0 STA T1LL	: (COSTANTE DI RITARDO
0035	03AF	A9 C3 LDA #\$C3	: PER 50 MS)
0036	03B1	8D 05 A0 STA T1HC	:QUESTO AVVIA IL TIMER
0037	03B4	C6 F7 DEC COUNT	:DECREMENTA IL CONTEGGIO
			:DI VENTI
0038	03B6	D0 31 BNE EXIT	:ESCI SE NON SI È
			:ANCORÀ CONTATO FINO A VENTI
0039	03B8	A9 14 LDA #\$14	:ALTRIMENTI RIPRISTINA CONTEGGIO

Programma 4-2: Tempo del Giorno (Figura 4-37 del testo)

```

0040 03BA 85 F7      STA COUNT      :UN INTERO SECONDO È TRASCORSO
0041 03BC A9 01      LDA #01
0042 03BE 18         CLC
0043 03BF 65 F6      ADC SECS      :AGGIUNGI 1 A SEC
0044 03C1 85 F6      STA SECS
0045 03C3 C9 60      CMP #060      :VEDI SE SI È ARRIVATI A 60 SECONDI
0046 03C5 D0 22      BNE EXIT      :SE NO, ESCI
0047 03C7 A9 00      LDA #000      :ALTRIMENTI PONI SECONDI A 0
0048 03C9 85 F6      STA SECS
0049 03CB A9 01      LDA #01
0050 03CD 18         CLC
0051 03CE 65 F5      ADC MIN      :E SOMMA 1 AI MINUTI
0052 03D0 85 F5      STA MIN
0053 03D2 C9 60      CMP #060      :VEDI SE SI È RAGGIUNTO 60 MINUTI
0054 03D4 D0 13      BNE EXIT      :SE NO, ESCI
0055 03D6 A9 00      LDA #000
0056 03D8 85 F5      STA MIN      :ALTRIMENTI PONI I MINUTI A 0
0057 03DA A9 01      LDA #01
0058 03DC 18         CLC
0059 03DD 65 F4      ADC HOUR     :E SOMMA 1 AD HOUR
0060 03DF 85 F4      STA HOUR
0061 03E1 C9 24      CMP #024      :VEDI SE SI SONO RAGGIUNTE 24 ORE
0062 03E3 D0 04      BNE EXIT      :SE NO, ESCI
0063 03E5 A9 00      LDA #000
0064 03E7 85 F4      STA HOUR     :ALTRIMENTI PONI HOUR A 0
0065 03E9 68         EXIT PLA      :RIMEMORIZZA LO STATO
0066 03EA 28         PLP
0067 03EB 40         RTI

```

ERRORS = 0000 < 0000 >

SYMBOL TABLE

SYMBOL VALUE

ACR	A00B	CLOCK	03A7	COUNT	00F7	EXIT	03E9
HOUR	00F4	INIT	0390	MIN	00F5	PLS	03EA
SECS	00F6	T1HC	A005	T1LL	A006		

END OF ASSEMBLY

*Programma 4-2: Tempo del Giorno (continua)*

LINE	#LOC	CODE	LINE
0002	0000		:QUESTA È UNA SEMPLICE ROUTINE DI CONTROLLO DOMESTICO
0003	0000		:CHE OPERA CON UN CICLO. AL TEMPO OPPORTUNO
0004	0000		:ESSA SALTA AD UN CERTO NUMERO DI UTENTI
			:DI SUBROUTINE
0005	0000		:ESEMPI
0006	0000		:DI DISPOSITIVI DI SERVIZIO:
0007	0000		:1) UNA SUBROUTINE POTREBBE CONTROLLARE IL TEMPO CORRENTE ED
0008	0000		: ACCENDERE UNA LUCE IN UN CERTO ISTANCE.
0009	0000		:2) UNA SUBROUTINE POTREBBE OSSERVARE LO STATO DI UN
0010	0000		: SISTEMA D'ALLARME E PRENDERE OPPORTUNI PROVVEDIMENTI
0011	0000		: SE SI RIVELA UN INTRUSO
0012	0000		DDRB = \$AC02
0013	0000		IORB = \$AC00
0014	0000		HOUR = \$00F4
0015	0000		MIN = \$00F5
0016	0000		OUTBYT = \$82FA
0017	0000		SCAND = \$8906
0018	0000		* = \$0200
0019	0200	D8	CONTRLCLD
0020	0201	A9 OF	LDA #\$0F
0021	0203	8D 02 AC	STA DDRB
			:PREDISPONI IL REGISTRO
			:DI DIREZIONE DATI
			:COME USCITA PER I RELÉ
0022	0206	A9 00	LDA #\$00
0023	0208	8D 00 AC	STA IORB
0024	020B	A5 F4	LOOP LDA HOUR
			:DISABILITA I RELÉ
			:QUESTO È IL CICLO
			:DI CONTROLLO PRINCIPALE
0025	020D	20 FA 82	JSR OUTBYT
			:INVIA AL DISPLAY
			:LE ORE ATTUALI
0026	0210	A5 F5	LDA MIN
0027	0212	20 FA 82	JSR OUTBYT
			:INVIA AL DISPLAY
			:I MINUTI ATTUALI
0028	0215	20 06 89	JSR SCAND
			:AZIONA IL DISPLAY
			:CON IL TEMPO
0029	0218	EA	.BYTE \$EA, \$EA, \$EA
0029	0219	EA	
0029	021A	EA	
0030	021B	EA	.BYTE \$EA, \$EA, \$EA
0030	021C	EA	
0030	021D	EA	
0031	021E	EA	.BYTE \$EA, \$EA, \$EA
0031	021F	EA	
0031	0220	EA	
0032	0221	EA	.BYTE \$EA, \$EA, \$EA
0032	0222	EA	
0032	0223	EA	
0033	0224	EA	.BYTE \$EA, \$EA, \$EA
0033	0225	EA	
0033	0226	EA	
0034	0227	EA	.BYTE \$EA, \$EA, \$EA
0034	0228	EA	
0034	0229	EA	
0035	022A	EA	.BYTE \$EA, \$EA, \$EA
0035	022B	EA	

:QUI L'UTENTE PUÒ POSIZIONARE  
 SALTI ALLE SUBROUTINE  
 :PER DISPOSITIVI  
 DI SERVIZIO

Programma 4-3: Controllo Domestico (Figura 4-38 del testo)

```

0035 022C EA
0036 022D EA      .BYTE $EA, $EA, $EA
0036 022E EA
0036 022F EA      .BYTE $EA, $EA, $EA
0037 0230 EA      .BYTE $EA, $EA, $EA
0037 0231 EA      .BYTE $EA, $EA, $EA
0037 0232 EA      .BYTE $EA, $EA, $EA
0038 0233 EA

```

} :QUI L'UTENTE PUÒ POSIZIONARE  
 } SALTI ALLE SUBROUTINE  
 } :PER DISPOSITIVI  
 } DI SERVIZIO

```

0038 0234 EA
0038 0235 EA
0039 0236 4C 0B 02    JMP  LOOP
0040 0239

```

ERRORS = 0000 < 0000 >

SYMBOL TABLE

SYMBOL VALUE

CONTRL	0200	DDRD	AC02	HOUR	00F4	IORB	AC00
LOOP	020B	MIN	00F5	OUTBYT	82FA	SCAND	8906

END OF ASSEMBLY

*Programma 4-3: Controllo Domestico (continua)*

LINE	LOG	CODE	LINE
0002	0000		:QUESTO È UN PROGRAMMA CHE COMPONE
0003	0000		:NUMERI TELEFONICI PRE-MEMORIZZATI. ESSO PRODUCE UN'USCITA
0004	0000		:A DUE TONI SU UN ALTOPARLANTE CONNESSO IN CONFIGURAZIONE 2
0005	0000		: (2 TONI - VEDI ALTOPARLANTE). QUESTI TONI RIPRODURRANNO
0006	0000		:IL TOCCO TELEFONICO PONENDO L'ALTOPARLANTE IN
0007	0000		:CORRISPONDENZA DELLA CORNETTA LATO BOCCA.
0008	0000		:PER IMPIEGARE IL PROGRAMMA CARICA IL NUMERO (1)
0009	0000		:TELEFONICO DOVUNQUE IN MEMORIA. UNA CIFRA PER BYTE
0010	0000		:TERMINANDO CON 0F (ESADEC.). PER ES. IL NUM.
0011	0000		:555-1212 RISULTEREBBE IN MEMORIA 05 05 05 01 02 01 02 0F (TUTTI ESADEC.).
0012	0000		:CARICANDO L'INDIRIZZO DEL NUMERO SI INTRODUCA
0013	0000		:PRIMA IL BYTE BASSO, NELLE LOC. 00C0 E 00C1.
0014	0000		:INOLTRE SI ENTRA A QUESTA SUB.
			:DA MONITOR O JSR AD ESSA DA UN ALTRO PROGRAMMA.
0015	0000		NUMPTR = \$00C0 :QUESTO PUNTA ALL'INDIRIZZO
			:DEL NUMERO TELEFONICO
0016	0000	ONDEL = \$40	:QUESTA È LA COSTANTE
			:DI RITARDO PER IL TEMPO
0017	0000	OFFDEL = \$20	:COSTANTE DI RITARDO PER IL TEMPO
			:QUANDO I TONI SONO 0
0018	0000	DELCON = \$FF	:COSTANTE DI RITARDO
			:PER SCOPI GENERALI
0019	0000	ACR1 = \$A00B	:QUESTI SONO I REGISTRI
			:TIMER MODE (TIMER 1)
0020	0000	ACR2 = \$AC0B	: (TIMER 2)
0021	0000	T1CH = \$A005	:QUESTO È IL CONTATORE DEL TIMER 1
			: (BYTE ELEVATO)
0022	0000	T1LH = \$A007	:LATCH DEL TIMER 1 (BYTE ELEVATO)
0023	0000	T1LL = \$A004	: (BYTE BASSO)
0024	0000	T2CH = \$AC05	:PER TIMER 2 ANALOGAMENTE A TIMER 1
0025	0000	T2LH = \$AC07	
0026	0000	T2LL = \$AC04	
0027	0000	* = \$0300	
0028	0300	A0 00 PHONE LDY #\$00	:INDICE PER LE CIFRE
			:DEL NUMERO TELEFONICO
0029	0302	B1 C0 DIGIT LDA (NUMPTR)Y	:ACCETTA CIFRA
0030	0304	C8 INY	
0031	0305	C9 0F CMP #\$0F	:VEDI SE FINE
			:DEL NUMERO TELEFONICO
0032	0307	D0 01 BNE NOEND	
0033	0309	60 RTS	:RITORNO SE SI (AL
			:MONITOR O
			:PROGRAMMA CHIAMANTE)
0034	030A	0A EA EA NOEND ASL A	:MULTIPLICA IL NUMERO PER
			:QUATTRO ALLA TABELLA INDICE
0035	030D	0A EA EA ASL A	: (OGNI INGRESSO ALLA TABELLA
			: È DI QUATTRO BYTE)
0036	0310	AA TAX	:X = INDICE DELLA TABELLA
0037	0311	A9 C0 LDA #\$C0	

Programma 4-4: Combinatore Telefonico (Figura 4-41 del testo)

0038	0313	8D 0B A0	STA ACR1	:PONI IL MODO DEL TIMER :A FREE RUNNING SU ENTRAMBI I TIMER
0039	0316	8D 0B AC	STA ACR2	
0040	0319	BD5D 03	LDA TABLE.X	:ACCETTA IL BYTE DI BASSO :ORDINE DEL PRIMO TONO
0041	031C	8D 04 A0	STA T1LL	:MEMORIZZA IL TIMER 1
0042	031F	E8	INX	
0043	0320	BD5D 03	LDA TABLE.X	:ACCETTA IL BYTE DI ORDINE :ELEVATO DEL PRIMO TONO
0044	0323	8D 07 A0	STA T1LH	:MEMORIZZA IL TIMER 1
0045	0326	8D 05 A0	STA T1CH	:QUESTO AVVIA :IL TIMER 1
0046	0329	E8	INX	
0047	032A	BD5D 03	LDA TABLE.X	:ACCETTA IL BYTE DI BASSO :ORDINE DEL SECONDO TONO
0048	032D	8D 04 AC	STA T2LL	:MEMORIZZALO NEL TIMER 2
0049	0330	E8	INX	
0050	0331	BD5D 03	LDA TABLE.X	:ACCETTA IL BYTE DI ORDINE :ELEVATO DEL SECONDO TONO
0051	0334	8D 07 AC	STA T2LH	:MEMORIZZALO NEL TIMER 2
0052	0337	8D 05 AC	STA T2CH	:QUESTO AVVIA :IL TIMER 2
0053	033A	A2 40	LDX #ONDEL	:ACCETTA LA COSTANTE DI :RITARDO DI TONO ACCESO
0054	033C	20 55 03 ON	JSR DELAY	:RITARDO MENTRE IL TONO È ACCESO
0055	033F	CA	DEX	
0056	0340	D0 FA	BNE ON	
0057	0342	A9 00	LDA #\$00	
0058	0344	8D 0B A0	STA ACR1	:DISABILITA ENTRAMBI I TIMER
0059	0347	8D 0B AC	STA ACR2	
0060	034A	A2 20	LDX #OFFDEL	:ACCETTA LA COSTANTE DI RITARDO :DI TONO SPENTO
0061	034C	20 55 03 OFF	JSR DELAY	:RITARDO MENTRE IL TONO È SPENTO
0062	034F	CA	DEX	
0063	0350	D0 FA	BNE OFF	
0064	0352	4C 02 03	JMP DIGIT	:RITORNA INDIETRO PER LA CIFRA :SUCCESSIVA DEL NUMERO TELEFONICO
0065	0355	:		
0066	0355	:		:QUESTA È UNA SEMPLICE ROUTINE DI RITARDO :PER L'ACCENSIONE E SPEGNIMENTO DEL TONO
0067	0355	:		
0068	0355	A9 FF	DELAY LDA #DELCON	:ACCETTA LA COSTANTE
0069	0357	38	WAIT SEC	:DI RITARDO DI QUESTA LUNGHEZZA
0070	0358	E9 01	SBC #\$01	
0071	035A	D0 FB	BNE WAIT	
0072	035C	60	RTS	
0073	035D	:		
0074	035D	:		:QUESTA È UNA TABELLA DELLE COSTANTI PER LE FREQUENZE
0075	035D	:		:DEL TONO PER OGNI CIFRA TELEFONICA. LE COSTANTI
0076	035D	:		:SONO LUNGHE 2 BYTE. PRIMA QUELLO BASSO.
0077	035D	:		

*Programma 4-4: Combinatore Telefonico (continua)*



```

0078 035D 13      TABLE .BYTE$13, $02, $76, $01 ;DUE TONI PER '0'
0078 035E 02
0078 035F 76
0078 0360 01
0079 0361 CD      .BYTE$CD, $02, $98, $01 ;DUE TONI PER '1'
0079 0362 02
0079 0363 9E
0079 0364 01
0080 0365 CD      .BYTE$CD, $02, $76, $01 ; '2'
0080 0366 02
0080 0367 76
0080 0368 01
0081 0369 CD      .BYTE$CD, $02, $53, $01 ; '3'
0081 036A 02
0081 036B 53
0081 036C 01
0082 036B 89      .BYTE$89, $02, $9E, $01 ; '4'
0082 036E 02
0082 036F 9E
0082 0370 01
0083 0371 89      .BYTE$89, $02, $76, $01 ; '5'
0083 0372 02
0083 0373 76
0083 0374 01
0084 0375 89      .BYTE$89, $02, $53, $01 ; '6'
0084 0376 02
0084 0377 53
0084 0378 01
0085 0379 4B      .BYTE$4B, $02, $9E, $01 ; '7'
0085 037A 02
0085 037B 9E
0085 037C 01
0086 037D 4B      .BYTE$4B, $02, $76, $01 ; '8'
0086 037E 02
0086 037F 76
0086 0380 01
0087 0381 4B      .BYTE$4B, $02, $53, $01 ; '9'
0087 0382 02
0087 0383 53
0087 0384 01
0088 0385          :FINE
ERRORS 0000 < 0000 >
SYMBOL TABLE
SYMBOL VALUE
ACR1      A00B      ACR2      AC0B      DELAY     0355      DELCON    00FF
DIGIT     0302      NOEND     030A      NUMPTR    00C0      OFF        034C
OFFDEL    0020      ON        033C      ONDEL     0040      PHONE     0300
T1CH      A005      T1LH      A007      T1LL      A004      T2CH      AC05
T2LH      AC07      T2LL      AC04      TABLE    035D      WAIT      0357
END OFF ASSEMBLY

```

*Programma 4-4: Combinatore Telefonico (continua)*

# APPENDICE D

## TABELLA DI CONVERSIONE ESADECIMALE

ES.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
ES.	DEC	ES.	DEC	ES.	DEC	ES.	DEC	ES.	DEC	ES.	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

## APPENDICE E

### TABELLA DI CONVERSIONE ASCII

ESAD.	BIT	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	-	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB		7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	--
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
	1111	SI	US	/	?	O	←	o	DEL

#### I SIMBOLI ASCII

NUL — Nullo	DLE — Perdita Collegamento Dati
SOH — Inizio della testata	DC — Controllo dispositivo
STX — Inizio del Testo	NAK — Riconoscimento negativo
ETX — Fine del Testo	SYN — Sincronismo non operativo
EOT — Fine della trasmissione	ETB — Fine del blocco di trasmissione
ENQ — Domanda	CAN — Cancella
ACK — Riconoscimento	EM — Fine del mezzo
BEL — Campana	SUB — Sostituto
BS — Spazio posteriore	ESC — Perdita
HT — Tabulazione orizzontale	FS — Separatore di file
LF — Incremento di riga	GS — Separatore di gruppo
V — Tabulazione verticale	RS — Separatore di record
FF — Incremento di scheda	US — Separatore di unità
CR — Ritorno carrello	SP — Spazio (Bianco)
SO — Sposta fuori	DEL — Cancella
SI — Sposta dentro	

## APPENDICE F

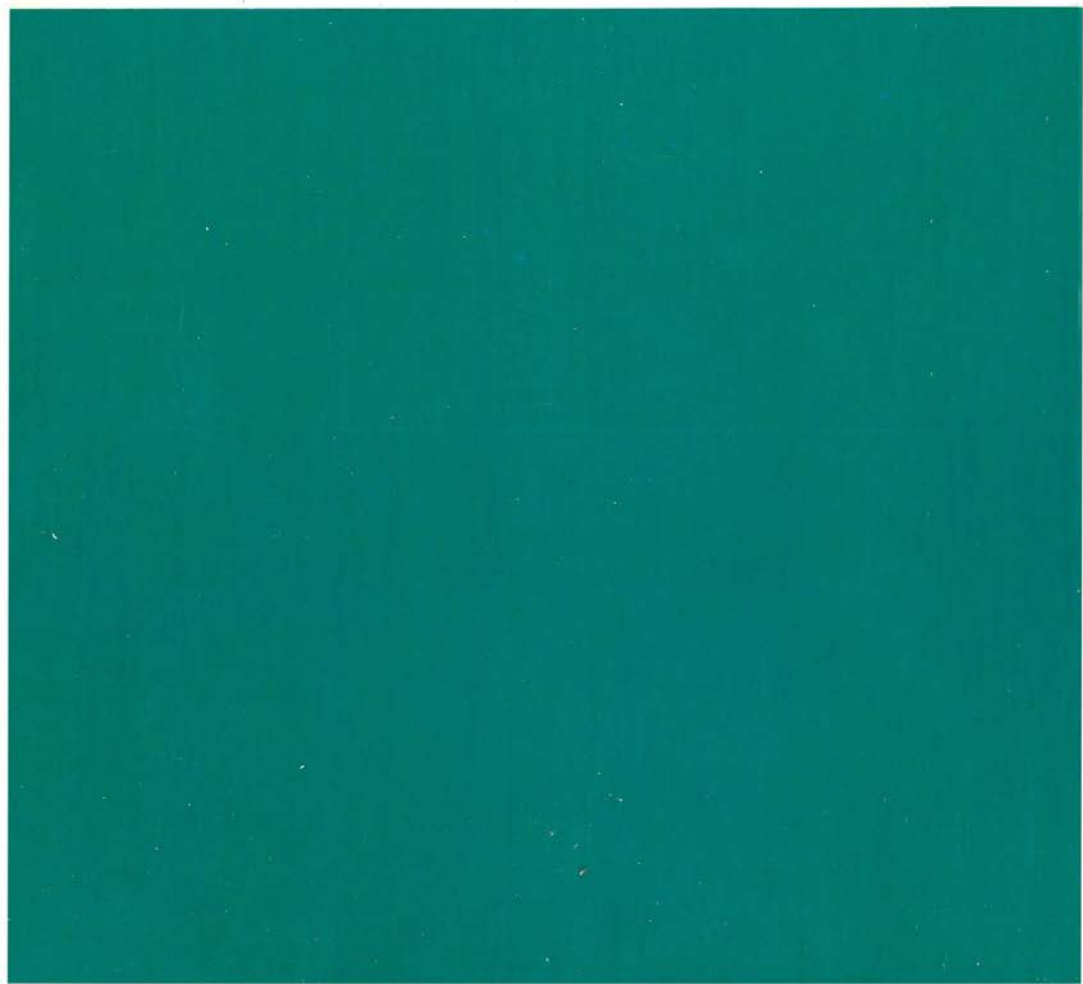
### ISTRUZIONI DEL 6502 (IN ORDINE ALFABETICO)

<b>ADC</b>	Somma con riporto	<b>JSR</b>	Salta alla subroutine
<b>AND</b>	AND Logico	<b>LDA</b>	Carica l'accumulatore
<b>ASL</b>	Spostamento Aritmetico a Sinistra	<b>LDX</b>	Carica X
<b>BCC</b>	Opera diramazione se carry è zero	<b>LDY</b>	Carica Y
<b>BCS</b>	Opera diramazione se carry è uno	<b>LSR</b>	Spostamento logico a destra
<b>BEQ</b>	Opera diramazione se risultato = 0	<b>NOT</b>	Non opera
<b>BIT</b>	Verifica di bit	<b>ORA</b>	OR Logico
<b>BMI</b>	Opera diramazione se negativo	<b>PHA</b>	Introduce A
<b>BNE</b>	Opera diramazione se diverso da 0	<b>PHP</b>	Introduce lo stato P
<b>BPL</b>	Opera diramazione se positivo	<b>PLA</b>	Estrae A
<b>BRK</b>	Break	<b>PLP</b>	Estrae lo stato P
<b>BVC</b>	Opera diramazione se overflow è 0	<b>ROL</b>	Rotazione a sinistra
<b>BVS</b>	Opera diramazione se overflow è 1	<b>ROR</b>	Rotazione a destra
<b>CLC</b>	Azzera carry	<b>RTI</b>	Ritorno da Interrupt
<b>CLD</b>	Azzera il flag decimale	<b>RTS</b>	Ritorno da subroutine
<b>CLI</b>	Azzera la disabilitazione interrupt	<b>SBC</b>	Sottrae con riporto
<b>CLV</b>	Azzera overflow	<b>SEC</b>	Pone carry ad 1
<b>CMP</b>	Confronta con l'accumulatore	<b>SED</b>	Pone decimale ad 1
<b>CPX</b>	Confronta con X	<b>SEI</b>	Pone disabilitazione interrupt ad 1
<b>CPY</b>	Confronta con Y	<b>STA</b>	Immagazzina l'accumulatore
<b>DEC</b>	Decrementa la memoria	<b>STX</b>	Immagazzina X
<b>DEX</b>	Decrementa X	<b>STY</b>	Immagazzina Y
<b>DEY</b>	Decrementa Y	<b>TAX</b>	Trasferisce A in X
<b>EOR</b>	OR Esclusivo	<b>TAY</b>	Trasferisce A in Y
<b>INC</b>	Incrementa la memoria	<b>TSX</b>	Trasferisce SP in X
<b>INX</b>	Incrementa X	<b>TXA</b>	Trasferisce X in A
<b>INY</b>	Incrementa Y	<b>TXS</b>	Trasferisce X in SP
<b>JMP</b>	Salta	<b>TYA</b>	Trasferisce Y in A



L. 13.500

Cod. 504 B





GRUPPO  
EDITORIALE  
JACKSON

Rodney Zaks

48