Guida alla programmazione in Assembler Z 80

sul PICO COMPUTER

Dante Del Corso

GRUPPO
EDITORIALE
JACKSON

Guida alla programmazione in Assembler Z 80

sul PICO COMPUTER

Dante Del Corso

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura del volume le signore Francesca di Fiore, Rosi Bozzolo e l'Ing. Roberto Pancaldi.

Prima edizione: 1982

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

PREFAZIONE

Questo breve testo è una guida introduttiva alla programmazione assembler attraverso una progressione di esercizi. Il calcolatore utilizzato è il "PICOCOMPUTER", già presentato in una serie di articoli su BIT (n. 3 - 4 - 5 -6 e 7). Esso è un sistema minimo completo di interfaccia di operatore e di memoria di massa, ed impiega il microprocessore Z80.

Non viene volutamente fornita una descrizione generale dello Z80 per il quale già esistono dettagliati manuali su architettura e istruzioni, nè dei "sistemi a microprocessore", argomento sul quale è disponibile un notevole assortimento di validi testi, citati nel seguito. Si presuppone che il lettore abbia una conoscenza di massima dell'organizzazione di un calcolatore (ad esempio cosa è una CPU od una memoria), e delle basi dell'algebra di Boole (AND, NOT, OR). I vari concetti sono comunque ripresi ed approfonditi con la progressione di esercizi.

I programmi qui riportati possono essere facilmente adattati ad altri sistemi Z80 od 8080 di caratteristiche analoghe al PICO, cioè dotati di tastiera e display esadecimali. Possono anche servire da traccia per una progressione analoga su macchine che utilizzano altre unità centrali (ad esempio RCA 1802, Philips 2650, Motorola 6800, i vari 65XX ecc.), ma in tal caso è necessaria una revisione sostanziale che tenga conto della diversa struttura della CPU.

Di ogni programma viene fornito il listato completo, e quindi non occorre disporre di assemblatori o altri supporti di sviluppo oltre al PICO stesso o piastra equivalente. Per eseguire gli esercizi consigliati (di cui non è fornito il listato) occorre peraltro "assemblare a mano" alcuni brevi programmi. Ciò va evidentemente considerato un ulteriore utile esercizio e non certo la proposta di un metodo di lavoro; rientra comunque nella impostazione globale di queste note che non presentano i fondamenti della "computer science", ma solo bit, registri, uni e zeri.

Questi esercizi, unitamente al Picocomputer o altra piastra analoga, sono in definitiva un punto di partenza per chi vuole fare una prima puntata nel mondo dei micro, con investimenti limitati e mantenendo sin dall'inizio l'integrazione hardware-software necessaria in questo campo.

È utile affiancare a questo testo, come riferimenti per approfondire ed espandere gli argomenti affrontati negli esercizi:

a) "Z80: Programmazione in linguaggio Assembly" di L. A. Leventhal Edizione Italiana Gruppo Editoriale Jackson.

Questo testo illustra in dettaglio la struttura dello Z80 e le varie istruzioni; riporta esempi di programmi (che possono essere implementati sul Picocomputer), e di circuiti periferici di interfaccia.

b) "Sistemi a microprocessore", di Dante Del Corso, Giordana e Pozzolo; Editore Boringhieri.

Descrive le organizzazioni generali di sistemi a microprocessori, partendo dalle famiglie logiche usate. Sono anche analizzate le diverse tecniche di interfacciamento per memorie e periferici.

SOMMARIO

CAPITOLO 1

SISTEMA PICOCOMPUTER

Descrizione generale

Il sistema PICOCOMPUTER è un microcalcolatore autonomo e completo di periferici per l'interazione con l'operatore ed interfacce per collegamenti verso l'esterno.

Un programma di "monitor" residente sulla scheda gestisce la consolle di operatore, attraverso la quale l'utente può scrivere, controllare e fare eseguire i propri programmi.
Lo schema a blocchi è in fig. 1-1.

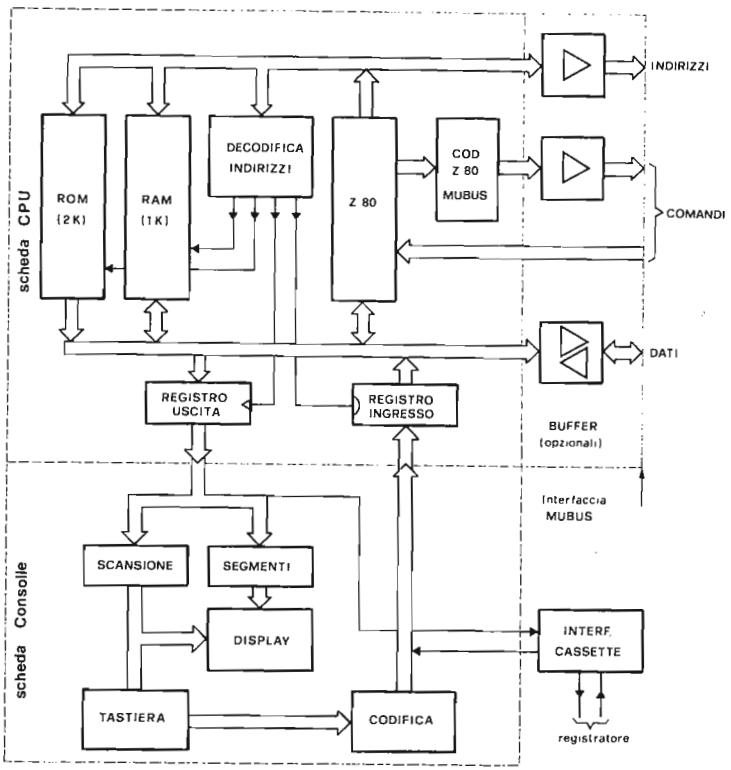


Figura 1.1 - Schema a blocchi del PICOCOMPUTER.

Il PICOCOMPUTER nella versione base comprende una scheda di unità centrale (CPU) e la scheda consolle d'operatore. Come indicato in fig. 1-1, la scheda CPU comprende anche memoria e registri di I/O, indirizzati come memoria. Questo insieme (RAM, ROM e registri) occupa complessivamente un banco di 4kbyte, posizionati dall'indirizzo 0000_H* a $0FFF_H$, come in fig. 1-2.

La scheda contiene una interfaccia completa verso il bus standard MUBUS, completamente bufferata, che permette di espandere il sistema con altra memoria e periferici. Questo manuale tratta in particolare l'uso del PICO come Single - Board - Computer; alcune possibili espansioni sono indicate nelle appendici.

Indirizzi		Memoria	Funzione
0000 – 03FF	1024 byte	EPROM 2716 (2K byte)	Programma di Monitor per gestione consolle (1K)
0400 – 07FF			Disponibile per programmi utente su EPROM (1K)
0800 – 0B7F		RAM 2 × 2114 (1K byte)	Disponibili per programmi di utenti
0B80 – 0BFF			Stack e variabili del monitor
0C00 – 0FFF	1K byte	Registri di I/O	Interfaccia di I/O per la consolle e per utente

Figura 1.2 - Mappa memoria e registri.

La scheda consolle d'operatore comprende un visualizzatore (display) per caratteri esadecimali ed una tastiera, il cui uso è descritto nel seguito. Questi periferici sono gestiti dalla unità centrale utilizzando parte dei registri di I/O presenti sulla scheda CPU.

Uso della consolle d'operatore

La memoria PROM contiene un programma che permette l'interazione con l'operazione attraverso la consolle (Programma di "Monitor"). Per iniziare l'esecuzione del

* Qui e nel seguito i numeri sono rappresentati in codice esadecimale.

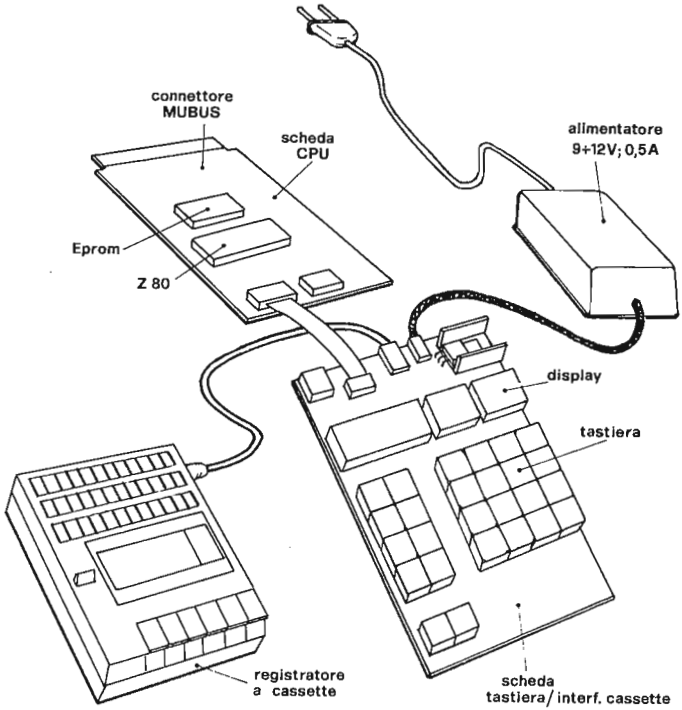


Figura 1.3 - Vista d'insieme e parti del PICOCOMPUTER.

monitor è sufficiente premere il pulsante di RESET, in quanto il programma inizia dalla cella 0000.

Questo capitolo descrive le funzioni del monitor Picomon V1, che impiega come consolle un display a otto cifre ed una tastiera a 24 tasti. Display e tastiera sono raggruppati nella piastra di controllo come indicato in Fig. 1.4.

Il visualizzatore di 8 cifre (Fig. 1.5) comprende:

- un campo "indirizzi" (ADD), in cui compare, in codice esadecimale, il puntatore di indirizzo generato dal monitor. (4 cifre).

- un campo "dati" (DATA), in cui compare il contenuto della cella indirizzata dal puntatore (campo ADD). (2 cifre).

- un campo "tastiera" (KEY) in cui compaiono, con scalamento automatico a sinistra, le ultime due cifre esadecimali introdotte dal campo dati della tastiera.

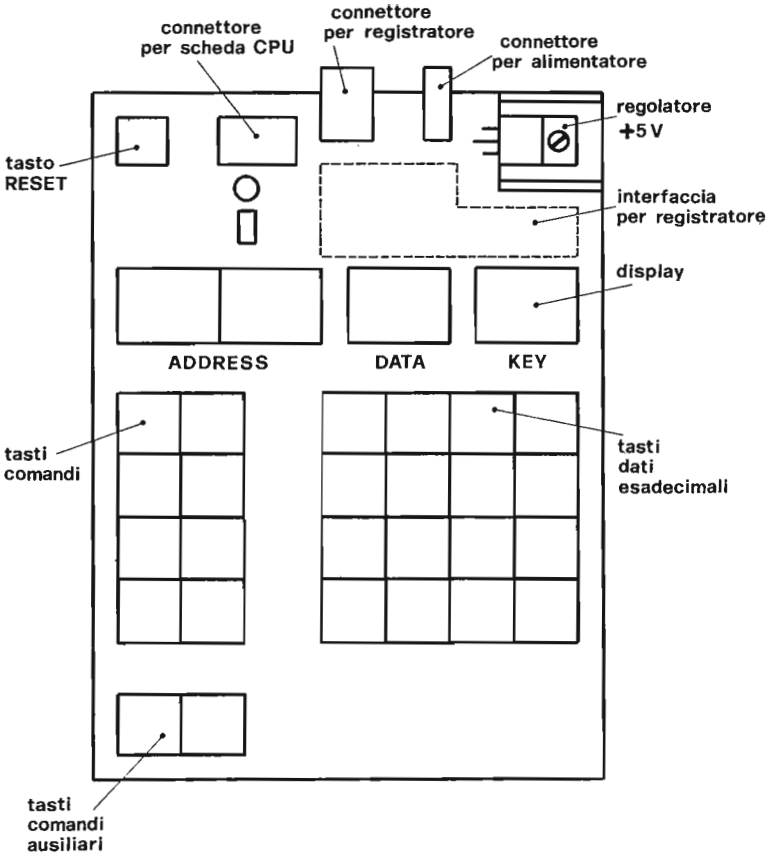


Figura 1.4 - Scheda tastiera/display.

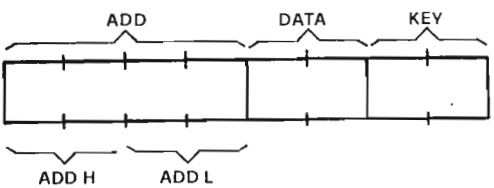


Figura 1.5 - Campi del display.

La tastiera (Fig. 1.6) comprende:

- un campo "dati (16 tasti da 0 a F). Il carattere introdotto premendo uno di questi tasti compare nella cifra di destra del campo KEY sul display. La cifra precedente viene scalata a sinistra.
- un campo "comandi" (8 tasti sulle due file a sinistra). Premendo uno di questi tasti si determina l'esecuzione di particolari funzioni da parte del programma di monitor.

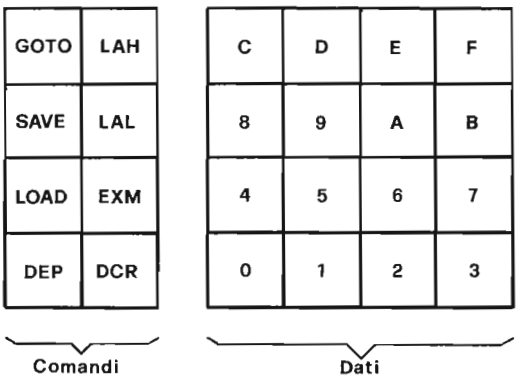


Figura 1.6 - Mappa della tastiera

Durante l'esecuzione del programma di Monitor il visualizzatore presenta sempre il contenuto della cella ADD, come indicato in Fig. 1.7. Tale cella è detta "aperta".

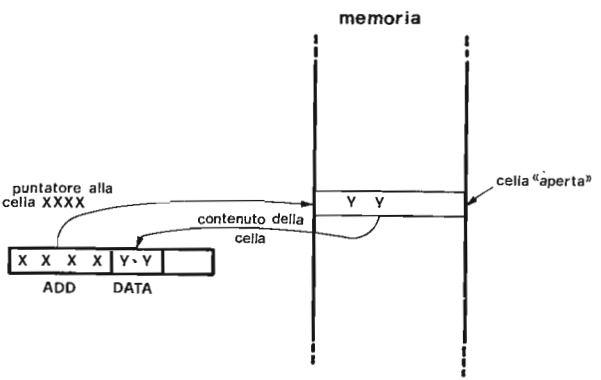


Figura 1.7 - Apertura di una cella di memoria.

Comandi del monitor Picomon V1

Questo programma di monitor consente di:

- leggere e scrivere su celle di memoria;
- eseguire programmi di utente;
- inserire dei breakpoint nei programmi di utente;
- verificare, ed eventualmente modificare, dopo un breakpoint, i registri della CPU;
- salvare su cassette magnetiche, e rileggere, programmi e dati.

Le funzioni dei tasti della consolle sono:

LAH	(Load Adress High): carica il contenuto del campo KEY, cioè le ultime due cifre battute sul campo dati della tastiera, nel campo ADDH (due cifre piú significative del puntatore di indirizzo).
LAL	(Load Address Low): analogo a LAH; carica (KEY) in ADDL.

Questi due comandi predispongono il puntatore di indirizzo su una determinata cella. La cella individuata dal puntatore è detta "aperta" (in quanto è possibile esaminarne il contenuto). È possibile dare prima LAH e quindi LAL o viceversa.

EXM	(EXaMine): incrementa di una unità il puntatore per leggere il contenuto della cella successiva.
DEP	(DEPosit): scrive nella cella "aperta" il contenuto del campo KEY ed incrementa il puntatore (per visualizzare la cella successiva).
DCR	(DeCRement): decrementa il puntatore; è utile per verificare il risultato di una operazione di DEP appena eseguita.

Questo gruppo di comandi del puntatore permette di scrivere programmi di utente nella memoria RAM e di verificare la corretta memorizzazione.

Il comando che controlla l'avvio dei programmi di utente è:

GOTO	Determina l'esecuzione del programma che inizia alla cella aperta (cioè all'indirizzo che compare nel campo ADD). Maggiori dettagli sul modo di funzionamento di questo comando sono contenuti nell'esercizio 2.3.

I comandi da usare per scrivere e rileggere dati e programmi su cassette magnetiche sono:

SAVE	Invia all'interfaccia per cassette magnetiche un messaggio comprendente il contenuto di una zona di memoria predefinita. Il messaggio comprende intestazioni e controlli di errore. La zona di memoria salvata è definita dal contenuto delle celle "FROM" e "TO"; il messaggio contiene i dati presenti da (FROM) a (TO) comprese. In coda al messaggio viene inserito un "autostart", cioè una intestazione speciale che indica dove riprendere l'esecuzione al termine della lettura. Questo indirizzo è contenuto nella cella "ABUF".

Prima di usare il comando SAVE occorre predisporre, usando i comandi definiti in precedenza, il contenuto delle celle:

FROM indirizzo iniziale : 0BFA/B_H
TO : indirizzo finale : 0BFC/D_H
ABUF: indirizzo di autostart : 0BFE/F_H

LOAD

Occorre scrivere prima il byte basso e poi quello alto di ciascun indirizzo.
Rilegge dalla interfaccia per cassette magnetiche un messaggio scritto mediante SAVE, verificando le intestazioni ed i controlli di errore. I dati riletti vengono scritti ove erano presenti in origine, e cioè nella zona di memoria da (FROM) a (TO). Al termine della lettura il controllo passa alla cella specificata come ABUF nel messaggio.
Eventuali errori rilevati in fase di lettura vengono segnalati sul visualizzatore presentando una stessa cifra in tutte le posizioni. Significato degli errori e maggiori dettagli sul funzionamento di questi programmi sono in appendice H.

L'uso delle cassette magnetiche richiede un modulatore / demodulatore, descritto anch'esso in appendice H. Un ulteriore comando a disposizione dell'operatore è quello di "breakpoint", o punto di interruzione della esecuzione.

Il breakpoint non è disponibile come comando diretto, ma l'operatore deve scrivere una istruzione di RST 3 (DF_H), usando normali comandi di consolle, nella cella ove si vuole interrompere l'esecuzione.

Quando, durante la esecuzione del programma di utente, si passa dalla cella ove è stato inserito il punto di interruzione, il controllo ritorna al programma di monitor ed i registri della unità centrale vengono trasferiti in una zona di memoria dove possono essere letti e modificati dall'operatore. Dopo un breakpoint il visualizzatore presenta, sul campo ADD, l'indirizzo ove è stata interrotta l'esecuzione.

Per riprendere l'esecuzione l'operatore deve ripristinare il contenuto originale della cella di breakpoint e dare un comando GOTO. Prima del comando GOTO è possibile modificare il contenuto dei registri semplicemente modificando le corrispondenti celle di memoria. Solo i registri del banco principale sono portati in memoria e possono quindi essere modificati.

La tabella indicante le celle corrispondenti a ciascun registro è in Fig. 1.8.

Stack Pointer	0BC 0/1	(Low/high)
IY	0BC 2/3	(Low/high)
IX	0BC 4/5	(Low/high)
HL	0BC 6/7	(L/H)
DE	0BC 8/9	(E/D)
BC	0BC A/B	(C/B)
AF	0BC C/D	(F/A)
Program counter	0BC E/F	(Low/high)

Figura 1.8 - Tabella area di salvataggio registri

Dato che il breakpoint si ottiene sostituendo il contenuto di una cella con la istruzione RST.3, è evidentemente possibile inserire punti di interruzione solo sulle celle di RAM che contengono il primo byte di una istruzione. È anche possibile inserire piú di un breakpoint, avendo cura di ripristinare tutte le celle per riprendere la normale esecuzione.

Uso del display e della tastiera nel programma di utente

Visualizzatore e tastiera corrispondono a due registri, rispettivamente a sola scrittura ed a sola lettura, allocati in memoria agli indirizzi da $0C00_H$ a $0FFF_H$. Nel seguito faremo riferimento alla sola posizione 0C00, anzichè a tutto il campo. La cella 0C00 viene indicata con l'etichetta CONADD (CONsolle ADDress). Il registro del display è organizzato come in Fig. 1.9.

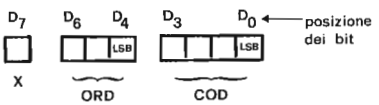


Figura 1.9 - Registro Display.

Il bit 7 è ignorato. Il campo ORD (bit 4—5—6) indica la posizione della cifra che si vuole illuminare (la cifra 0 è la prima a destra).

Il campo COD (bit 0—1—2—3) contiene la codifica binaria del carattere esadecimale che compare nella posizione ORD.

Ad esempio, se sul registro del display viene scritto 36 (oppure B6), comparirà un 6 in posizione 3 (Fig. 1.10).

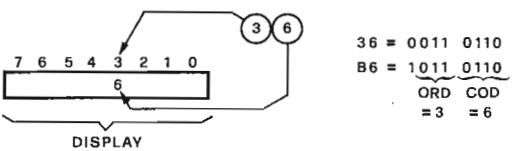


Figura 1.10

Il display può essere utilizzato dai programmi di utente, avendo cura di predisporre il formato dei dati secondo le regole esposte.

La tastiera è organizzata a matrice di righe (6) e colonne (4). La scansione seleziona ciascuna riga secondo i bit ORD del registro display (la riga 0 è quella piú in basso). Se viene premuto un tasto (appartenente alla riga Ri e alla colonna Cj), nel momento in cui è selezionata la riga Ri sui bit COL del registro tastiera compare il numero d'ordine

della colonna (j). Va contemporaneamente a 0 il bit 7 (indicazione di tasto premuto TPR) (Fig. 1.11).

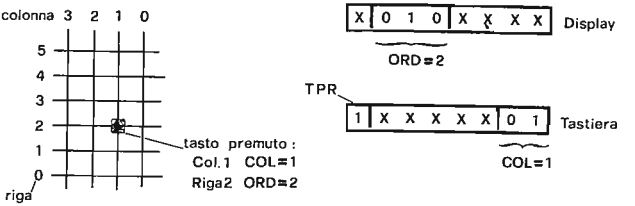


Figura 1.11

La codifica completa del tasto deve essere ricostruita, a software, utilizzando ORD e COL. Una spiegazione piú dettagliata della gestione di visualizzatore e display nel programma di monitor è nella appendice C.

Periferici disponibili per i programmi di utente

Rimangono disponibili e utilizzabili per i programmi di utente all'indirizzo $0C00_H$:

- *Registro di scrittura (output):*

tutti i bit sono utilizzati dal monitor per la gestione di visualizzatore, tastiera e cassette magnetiche. I bit $0 \div 3$ possono essere usati dal programma di utente rinunciando al visualizzatore. Il bit 7 non interferisce con la consolle, ma è usato come uscita per le cassette magnetiche. Usando i bit $4 \div 6$ si perde il controllo della tastiera.

- *Registro di lettura (input):*

Sono usati i bit:
0-1 codifica colonna tastiera;
6 ingresso per cassette magnetiche;
7 segnale di tasto premuto.
Tutti gli altri sono disponibili per programmi di utente: L'uso di questi registri è indicato in Fig. 1.12.

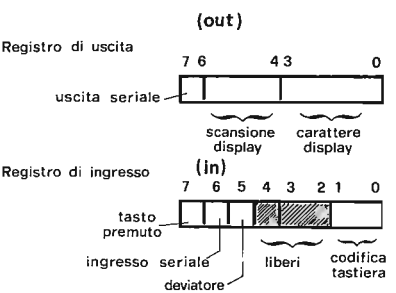


Figura 1.12 - Registri della consolle.

Piedinatura dei connettori sulla scheda CPU

A	Piedino	Funzione	
	1	OUT 4	
	2	OUT 5	
	3	OUT 6	
	4	OUT 7	
	5	OUT 0	
	6	OUT 1	
	7	OUT 2	
	8	MASSA	(GND)
	9	OUT 3	
	10	IN 1	
	11	IN 0	
	12	RESET	(RST*)
	13	IN 5	
	14	IN 6	
	15	IN 7	
	16	+5V	

B			
	1	IN 0	
	2	IN 1	
	3	IN 2	
	4	IN 3	
	5	IN 4	
	6	IN 5	
	7	IN 6	
	8	MASSA	(GND)
	9	IN 7	
	10	OUT 4	
	11	OUT 5	
	12	OUT 6	
	13	OUT 7	
	14	PUL 1	
	15	PUL 2	
	16	+5V	

Figura 1.13 - Connettori di I/O della scheda CPU.

Uso del buffer di display

I caratteri esadecimali presentati sul display corrispondono al contenuto delle celle di memoria $0BF0/1/2/3_H$ (Buffer di display DISBUFF), come indicato in Fig. 1.14.

È quindi possibile far comparire caratteri esadecimali sul display semplicemente scrivendoli, nelle celle BF0/1/2/3 nel corso della esecuzione del programma di utente e ridando successivamente il controllo al monitor.

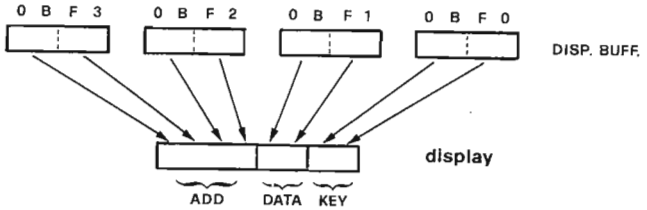


Figura 1.14 - Mappamento della memoria sul display sotto monitor.

Le celle 0BF3/2 ricopiano il contenuto dei registri DE, e la cella 0BF1 il contenuto della cella puntata da DE. La cella 0BF0 è utilizzabile direttamente; scrivendo in essa un dato e cedendo il controllo al monitor, il dato viene presentato nel campo KEY.

Avvertenza

Gli esercizi qui proposti usano la maggior parte delle istruzioni Z80 e comunque tutte quelle fondamentali o di uso piú comune.

Alcune istruzioni, il cui comportamento non può essere verificato mediante il solo PICOCOMPUTER non vengono mai usate (ad esempio quelle relative alle interruzioni modo 0 e 2).

Nessun esercizio usa i registri IX e IY, peraltro sostanzialmente analoghi al registro HL, usato come puntatore alla memoria. La tecnica che permette di passare dalle istruzioni relative ad HL a quelle per IX ed IY è descritta in appendice A. Questa stessa appendice riporta tutte le istruzioni Z80, comprese quelle che non sono mai state usate negli esercizi.

CAPITOLO 2

ESERCIZI

1. Uso della tastiera

Per prima cosa conviene impratichirsi nell'uso della consolle provando a scrivere e verificare sequenze di dati in memoria. Nel seguito il simbolo ... indica una operazione sulla tastiera. Nel caso di caricamento di una cifra esadecimale (campo dati della tastiera), viene usato il simbolo [#].

Le operazioni da eseguire per aprire una cella sono:

- Predisporre la parte alta del puntatore (due cifre nel campo dati): [#] [#]
- Caricare la parte alta del puntatore: [LAH]
- Predisporre la parte bassa del puntatore: [#] [#].
- Caricare: [LAL]

La cella di memoria individuata dal puntatore è ora "aperta" e sul campo dati del display compare il suo contenuto.

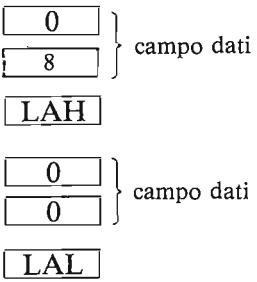
Ad esempio, aprendo la cella 0000 si legge "31", corrispondente al primo byte del monitor. È possibile esaminare le celle successive con [EXM] e tornare indietro con [DCR]; partendo dalla cella 0000 si ha:

ADD	DATA	Comando
0000	31	[EXM]
0001	9 F	[EXM]
0002	0 B	[EXM]
0003	2 2	[DCR]
0002	0 B	[EXM]
•		•
•		•
•		•

Se la locazione aperta è in memoria RAM (nel caso del Picocomputer da 0800_H a $0BFF_H$, di cui le celle $0B80_H$ $0BFF_H$ sono usate dal monitor), è anche possibile scrivere nella cella tramite il comando [DEP]

Ad esempio:

- Caricare il puntatore con 800:



A questo punto sul campo indirizzi del display comparirà 0800; nel campo dati compare il contenuto attuale della cella, che possiamo modificare digitando:

# #	nuovo dato da scrivere in 800;
DEP	il dato che compare nel campo KEY viene scritto in 800; il puntatore è automaticamente incrementato e passa alla locazione 801.

Per verificare il contenuto di 800 si può dare il comando DCR; il puntatore ritorna ad 800 e ricompare il dato precedentemente caricato. Conviene eseguire qualche esercizio di caricamento di sequenze di dati, con successiva verifica, in varie locazioni di memoria.

La sequenza 00, 11, 22 ... FF, che contiene tutte le combinazioni di bit, può essere usata per fare una rapida verifica della funzionalità della memoria RAM.

È anche possibile verificare che le operazioni di scrittura eseguite su memoria ROM (0000 - 07FF) o su memoria non esistente (1000 - FFFF), non danno alcun risultato.

Esercizio 1.1

Caricare la sequenza 00, 11, ...FF da 800 a 80F:

0	0	LAL	0	8	LAH	apertura cella 800
0	0	DEP				scrive 00 in 800
1	1	DEP				scrive 11 in 801
.	.	.				.
.	.	.				.
.	.	.				.
F	F	DEP				scrive FF in 80F

Verificare il corretto caricamento di questi dati: usando il comando DCR oppure riaprendo 800 (è sufficiente ricaricare 00 in ADDL).

Esercizio 1.2

Ripetere le operazioni dell'esercizio 1.1 su zone di memoria che non siano RAM:

- memoria ROM: da 0000_H a $07FF_H$
- memoria non esistente: da 1000_H a $FFFF_H$.

Verificare che nel caso di memoria ROM (a sola lettura) le operazioni di scrittura lasciano inalterato il contenuto; e che nel caso di memoria non esistente viene sempre letto FFH (le linee aperte vengono interpretate come 1 logico; quindi se alle linee dati non viene collegato nessun dispositivo il micro legge sempre FF_H.

Gli esercizi di programmazione che seguono sono suddivisi in vari gruppi; gli esercizi dei gruppi da 2 a 7 hanno lo scopo di familiarizzare il lettore con i comandi del monitor e con l'uso di determinate classi di istruzioni. Gli esercizi dei gruppi 8 e seguenti portano a realizzare brevi programmi completi finalizzati ad applicazioni specifiche.

Ad ogni gruppo di esercizi è premessa una spiegazione sintetica delle nuove istruzioni di volta in volta utilizzate; per maggiori dettagli su queste si rimanda al manuale Z80. I codici operativi son riportati oltre che nel manuale stesso, in appendice A. Tutti i programmi sono presentati nel formato usuale dei listati:

Indirizzo	Codice operativo	Etichetta	Mnemonici Assembler	Commento
(ADD)	(OPCODE)	(LABEL)	(ASSEMBLER)	(COMMENT)

2. Istruzioni di trasferimento dati

Le istruzioni di trasferimento determinano un passaggio di informazione tra un registro "sorgente" ed un registro "Destinazione" (Fig. 2.1).

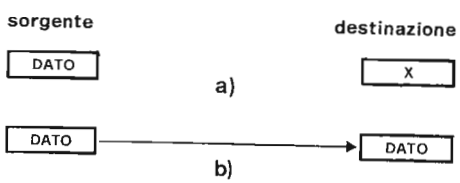


Figura 2.1 - Istruzioni di trasferimento a) prima; b) dopo l'esecuzione. Il contenuto precedente della destinazione (x) viene cancellato.

La struttura degli mnemonici assembler per la istruzione di trasferimento è sempre:

LD <destinazione>, <sorgente>

↓

Codice mnemonico (LoaD)

possono essere registri, celle di memoria, coppie di registri, valori immediati.

Solo alcune combinazioni sorgente/destinazione sono ammesse; per lo Z80 sono valide ad esempio le istruzioni:

LD <registro>, <registro>
LD <registro>, <valore immediato>
LD <cella di memoria>, <accumulatore>
LD <accumulatore>, <cella di mem. puntata da una coppia di registri>

ed altre.

Gli esercizi di questo capitolo presentano questa classe di istruzioni con alcuni esempi di uso appositamente studiati per il Picocomputer.

Per visualizzare il risultato di una operazione di trasferimento è possibile usare diverse tecniche:

a) leggere, usando i comandi del monitor, la cella di memoria destinazione (per trasferimenti verso la memoria);
b) leggere, dopo un breakpoint, il registro destinazione (trasferimenti tra registri);
c) usare, come destinazione, il registro di uscita che controlla il visualizzatore (indirizzo 0C00); in questo modo il dato viene presentato come indicato in Fig. 1.10.
d) scrivere il dato nel buffer del visualizzatore (celle 0BF0/1/2/3) e dare il controllo al monitor.

Esercizio 2.1

Trasferire una parola assegnata (xy in esadecimale) sul registro di display.

ADD	OPCODE	LABEL	ASSEMBLER	COMMENT
800	3E	E 2.1	LD A, "xy" (1)	Carica "xy" (due caratteri esadecimali)
801	xy			nel registro A
802	32		LD (0C00), A (2)	Trasferisce il contenuto di A (cioè xy) nello
803	00			indirizzo 0C00 (regi-
804	0C			stro di display)
805	76		HALT	Questa istruzione ferma l'esecuzione

(1) Questa è una istruzione del tipo: LD <registro>, <valore immediato> il registro è l'accumulatore A; il valore immediato è xy
(2) Istruzione del tipo: LD <cella di memoria> <accumulatore>

L'istruzione di HALT mette lo Z80 in uno stato di sospensione dell'esecuzione segnalato dalla CPU attivando il piedino "HALT". Questo è collegato al LED posto sulla scheda CPU tra i due connettori A e B, che pertanto si accende dopo l'esecuzione della istruzione di HALT. Per uscire dallo stato di HALT occorre dare un reset o una interruzione.

Si può verificare l'esecuzione di questo breve programma semplicemente osservando la cifra che compare sul display. Il formato è quello di Fig. 1.10.

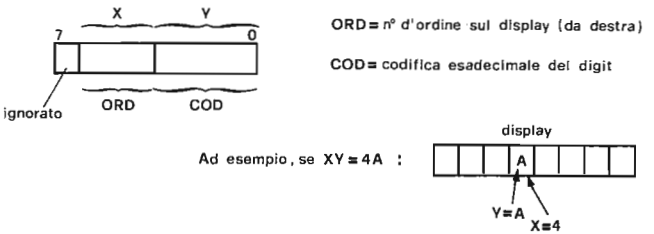


Figura 2.2

Le operazioni da eseguire per caricare il programma sono:
- Aprire la locazione 0800:

[0] [8] [LAH] [0] [0] [LAL]

A questo punto sul campo ADD compare 0800
- Caricare i codici delle istruzioni:

3E[1], [DEP]

"xy", [DEP] dato da trasferire sul display

3 2, [DEP]

0 0, [DEP]

0 C, [DEP]

7 6, [DEP]

Conviene verificare la corretta scrittura riaprendo 800:

00, [LAL]; [EXM]; [EXM]..

Per far eseguire il programma occorre riaprire 800 e dare il comando GOTO:

00, [LAL] [GOTO]

(1) Nel seguito l'introduzione di due caratteri nel campo KEY è indicata brevemente come # #, (anzichè [#] [#])

Sul display comparirà così solo il carattere "y" in posizione "x". Dato che il programma si chiude con un Halt, per tornare al monitor occorre dare un RESET. In HALT non è possibile usare la tastiera, perchè quando il sistema non sta eseguendo il monitor, essa è completamente muta.

Esercizi consigliati

- Ripetere il programma 2.1 con altri valori di xy, facendo comparire un carattere predeterminato in una posizione assegnata.

- Verificare che il bit 7 è ignorato (ad esempio 17 e 97 producono lo stesso effetto).

Esercizio 2.2

Per trasferire un dato sul registro del display è possibile usare anche un altro programma:

800 801 802	21 00 0C	E 2.2	LD HL, 0C00 (1)	Carica nella coppia di registri HL l'indirizzo del registro di display
803 804	36 "xy"		LD (HL), "xy" (2)	Trasferisce il dato immediato xy nella cella puntata da HL (cioè nel registro di display 0C00).
805	76		HALT	

La coppia di registri HL viene qui usata come "puntatore" per la cella destinazione. Per caricare e far eseguire questo programma seguire la procedura dell'esercizio 2.1:

- Apertura della cella 800
- Caricamento e verifica
- Comando GOTO

Esercizio 2.3

Già con questi esercizi è possibile verificare il funzionamento del breakpoint. Inserendo un breakpoint dopo una istruzione di caricamento di registri, possiamo controllare nella tabella dei registri (fig. 1.8) l'effettiva esecuzione dell'istruzione. La sequenza di operazioni eseguita dal microprocessore quando, durante l'esecuzione di un programma, viene incontrato il codice di breakpoint DF_H è indicata in figura 2.3.

(1) Istruzione del tipo LD <coppia di registri , <valore immediato>. La destinazione è una coppia di registri, e quindi il dato immediato è di 16 bit.

(2) Istruzione del tipo LD <cella puntata dalla coppia di registri HL> , <valore immediato>. il contenuto di HL è 0C00, ed il dato immediato viene trasferito in questa locazione, cioè sul registro consolle.

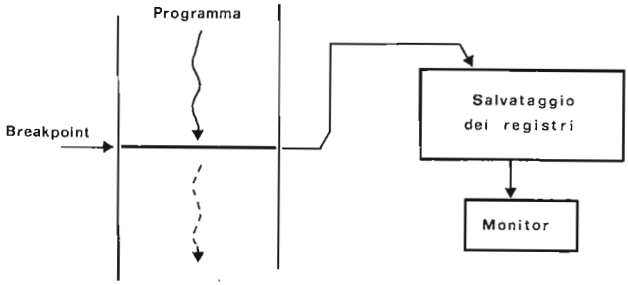


Figura 2.3 - Esecuzione di un breakpoint.

La procedura da seguire per inserire un breakpoint è:
a) aprire la cella in cui si vuole inserire il punto di interruzione;
b) caricare in essa il codice di breakpoint DF_H

Il comando GOTO opera in modo esattamente complementare inizializzando i registri con i valori contenuti nella tabella. (Figura 2.4)

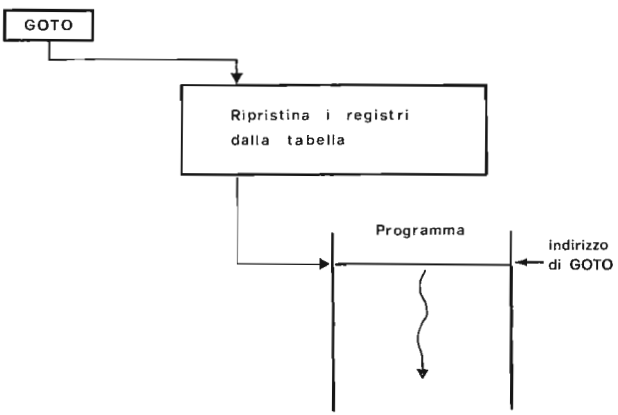


Figura 2.4 - Esecuzione di un GOTO.

Se l'operatore modifica, dopo un breakpoint, i valori della tabella, è possibile continuare l'esecuzione del programma con il nuovo valore nei registri. Questa procedura può essere applicata all'esercizio 2.1:

- caricare il programma (valore xy in A);
- mettere il breakpoint in 802 come indicato cioè caricare DF;
- comando GOTO a 800.

L'esecuzione si interrompe a 802 e questo indirizzo compare sul visualizzatore; aprendo la cella 0BCD compare il contenuto di A, (cioè "xy"). Modifichiamo il contenuto della cella - in "x_1y_1" (comando DEP). - Far proseguire l'esecuzione aprendo la cella 802, ripristinare il contenuto (32_H) e dare GOTO. Sul display compare il carattere corrispondente al dato modificato in A (cioè x_1y_1 e non xy).

Esercizio 2.4

Permette di verificare il contenuto del banco completo di registri:

800 802 804	3E, xx 06, yy 0E, zz	E 2.4	LD A, xx (1) LD B, yy (1) LD C, zz (1)	Caricamento di registri singoli	
806 809	11, ww, w'w' 21, kk,k'k'		LD DE, w'w'ww (2) LD HL, k'k'kk(2)	Caricamento di coppie di registri	(Notare la inversione dei byte)
80C	DF		RST 18	Breakpoint	

Grazie al breakpoint inserito in 80C, possiamo verificare nella tabella di Fig. 1.8 il contenuto dei vari registri.

Il contenuto dell'area di salvataggio è:

0BC 6	: kk	0BCA	: zz
0BC 7	: k' k'	0BCB	: yy
0BC 8	: ww	0BCC	: flag
0BC 9	: w' w'	OBCD	: xx
		0BCE/F	: 0C08 (PC al breakpoint)

Esercizio 2.5

Permette di verificare che il comando GOTO inizializza i registri della tabella:
- Caricare in 0BCD (registro A) xy
- Caricare in 0BC7/6 (registri H e L) 0C00 (CONADD)
- Caricare il programma:

800 801	77 76	E 2.5	LD (HL), A (3) HALT	Muove A nella cella puntata da HL.

(1) Istruzioni del tipo <registro>, <valore immediato di 8 bit>
(2) Istruzioni del tipo: LD <coppia di registri>, <valore immediato di 16 bit>
(3) Istruzioni del tipo: LD <cella puntata> da <coppia di registri> <accumulatore>

Dato il comando di esecuzione, sul display compare una cifra corrispondente alla codifica di xy come nell'esercizio 2.1. Infatti il comando GOTO inizializza i registri:

```
HL    a    CONADD    (0C00)
A     a    xy
```

ATTENZIONE: ricordare che il breakpoint:

- può essere inserito solo su memoria RAM;
- salva solo i registri del banco principale;
- va posto su una cella che contiene un codice operativo (I byte dell'istruzione).

e che il comando GOTO ripristina solo i registri del banco principale.

Esercizi consigliati

A questo punto, usando i metodi prima descritti per verificare il risultato delle operazioni, e cioè trasferimenti sul registro display o breakpoint, il lettore può scrivere programmi di poche istruzioni in cui siano utilizzate le diverse combinazioni sorgente / destinazione dello Z80.

Le combinazioni ammesse ed i relativi codici operativi sono nella Tabella I. Un assortimento di queste istruzioni è già stato usato negli esercizi precedenti.

Si può notare come certi registri (ad esempio l'accumulatore A) siano "privilegiati", in quanto ammettono una piú varia scelta di modi di indirizzamento.

Istruzioni di ingresso/uscita

Una classe particolare di istruzioni di trasferimento è formata dalle istruzioni di ingresso e di uscita (INPUT ed OUTPUT). Queste istruzioni permettono di trasferire dati di 8 bit da o verso 256 celle appartenenti ad uno "spazio di indirizzamento" differenziato ad hardware dalla normale memoria. Queste celle costituiscono i registri di ingresso ed uscita dei periferici.

Le istruzioni di ingresso/uscita sono:

OUT	<destinazione>	<sorgente>
↓	↓	↓
Codice assembler	può essere un valore immediato o il contenuto del registro C (seleziona 1 cella su 256)	può essere un registro, oppure la cella puntata da HL

e

IN	<destinazione>	<sorgente>
	↓	↓
	registro, oppure (HL)	valore immediato, o contenuto di C (seleziona 1 cella su 256)

Non vi sono esercizi su queste istruzioni perchè il Picocomputer non possiede, nella configurazione base, registri selezionati come periferici. Occorre però precisare che, per scambiare dati con l'esterno non è indispensabile usare le istruzioni IN e OUT.

Infatti i registri di interfaccia possono essere selezionati dallo spazio di indirizzamento dei periferici (256 celle attivate con le istruzioni IN e OUT nello Z80), oppure dallo spazio di indirizzamento della memoria normale, (2^{16} = 65.536 celle), usando le istruzioni di LOAD. In quest'ultimo caso si può riservare un blocco di celle per i registri di I/O (vedi Fig. 2.5) e si parla di periferici "mappati su memoria" (memory - mapped). I registri del PICO-computer che interfacciano tastiera e visualizzatore sono appunto celle mappate su memoria all'indirizzo $0C00_H$.

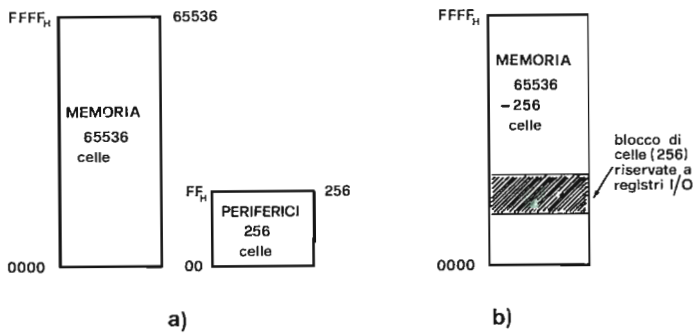


Figura 2.5 - Spazi di indirizzamento per memoria e registri di I/O a) separati, b) mappati su memoria.

Tabella I - Istruzioni di trasferimento (LoaD)

a) Tra registri di 8 bit n = > numero di 8 bit.

LD	B	C	D	E	H	L	(HL)	A	n sorgente
B	40	41	42	43	44	45	46	47	06,n
C	48	49	4A	4B	4C	4D	4E	4F	0E,n
D	50	51	52	53	54	55	56	57	16,n
E	58	59	5A	5B	5C	5D	5E	5F	1E,n
H	60	61	62	63	64	65	66	67	26,n
L	68	69	6A	6B	6C	6D	6E	6F	2E,n
(HL)	70	71	72	73	74	75	—	77	36,n
A	78	79	7A	7B	7C	7D	7E	7F	3E,n

← sorgente

↑

destinazione

b) Speciali per l'accumulatore A.

LD	→ A	← A
(BC)	0A	02
(DE)	1A	12
(HL)	7E	77
(nn)	3A,n,n	32,n,n
I	ED,57	ED,47
R	ED,5F	ED,4F

→ A : A è destinazione
← A : A è sorgente
n n = > numero di 16 bit

c) Per coppie di registri.

LD	← n,n	← (n,n)	→ (n,n)
BC	01,n,n	ED,4B, ,	ED,43,n,n
DE	11,n,n	ED,5B,n,n	ED,53,n,n
HL	21,n,n	2A,n,n	22,n,n
SP	31,n,n	ED,7B,n,n	ED,73,n,n

d) Istruzioni di I/O.

	B	C	D	E	H	L	A		
IN r,(c)	ED,40	ED,48	ED,50	ED,58	ED,60	ED,68	ED,78	IN A,(n)	DB,n
OUT (c),r	ED,41	ED,49	ED,51	ED,59	ED,61	ED,69	ED,79	OUT(n),A	D3,n

3. Istruzioni aritmetiche e logiche

Questo gruppo di esercizi consente di verificare l'esecuzione di operazioni logiche (AND, ON, EXOR) e aritmetiche (+, -) da parte del microprocessore.

I dati di partenza possono essere contenuti nei registri, in memoria o ancora forniti come valore immediato; il risultato è disponibile nell'accumulatore A (operazioni su dati di 8 bit) o nella coppia di registri HL (operazioni su dati di 16 bit). Lo schema di esecuzione è in Fig. 2.6.

Per visualizzare il risultato possiamo usare uno dei procedimenti descritti negli esercizi precedenti:

- trasferimento sul registro display
- lettura di A o HL dopo un breakpoint

- trasferimento sul buffer di display e ritorno al monitor.

Il risultato di una operazione logica o aritmetica influenza i bit del registro di stato secondo le modalità indicate nella tabella di pag. 25.

Per una descrizione dettagliata delle operazioni svolte da ciascuna istruzione di questa categoria si rimanda al manuale dello Z80.

Gli esercizi di questo capitolo presentano un assortimento di operazioni logiche ed aritmetiche, utilizzando le diverse tecniche di indirizzamento.

Il formato degli mnemonici assembler per le istruzioni loghiche aritmetiche è:

< codice del tipo di operazione> <operando 2>

Il primo operando è indicato implicitamente nel codice dell'operazione.

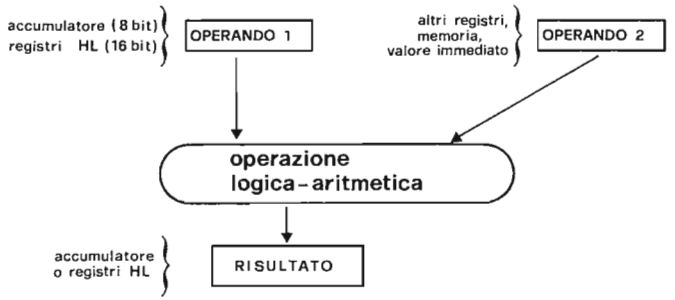


Figura 2.6 - Esecuzione delle istruzioni logico-aritmetiche

Esercizio 3.1

Somma di due dati XX e YY. Il primo addendo è in A; il secondo è fornito come valore immediato.

800	3E, xx	E 3-1	LD A, "xx"	Primo dato in A
802	C6, yy		ADD A, "yy"	Somma con il secondo (valore immediato)
804	76		HALT	

A questo punto il risultato xx + yy è in A; per visualizzare occorre inserire un breakpoint all'indirizzo 804, e leggere da monitor la cella 8ED, come per l'esercizio 2.3.

In alternativa, chiedendo il programma con:

804	32,00,0C		LD (0C00), A
807	76		HALT

Il contenuto di A viene portato sul registro display, e accende una cifra secondo il formato descritto nell'esercizio 2-1.

Ancora in alternativa, possiamo trasferire A sul buffer del display (vedi pag. 10), e precisamente sulla cella BF0 che corrisponde al campo KEY, e tornare al monitor.

804	32, F0, 0B		LD (0BF0), A	; trasferimento sul buffer display
807	C7		RST 0	;istruzione che ridà il controllo al monitor.

Il contenuto di A compare cosí, codificato in esadecimale, sul campo ADD_H.

Esercizi consigliati

- Ripetere l'esercizio 3-1 con altri dati, eseguendo altre operazioni (AND, OR, XOR, ...), verificando i risultati con i metodi sopra descritti.
- Eseguire catene di operazioni logiche e/o aritmetiche, verificando i risultati intermedi inserendo dei breakpoint. (ad esempio ADD A, xx; AND yy; OR zz....)

Esercizio 3.2

I risultati delle operazioni logico-aritmetiche modificano il contenuto del registro di stato (Flag). Anche questo registro può essere letto dopo un breakpoint (indirizzo BCC). Il formato del registro di stato è riportato nella figura 2.7.

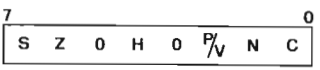


Figura 2.7 - Registro di stato.

Tabella Flag di stato

Flag	bit n°	Funzione
5	7	Segno: vale 1 se il contenuto di A è negativo. In pratica ripete il bit 7 dell'accumulatore (i numeri sono rappresentati in codice complemento a due, quindi quelli negativi hanno come bit piú significativo un 1).
Z	6	Zero: vale 1 se il contenuto di A è zero, o dopo istruzioni di "Compare" con esito positivo (vedi E 3.4)
H	4	Riporto intermedio (Half carry). Va ad 1 se c'è stato un riporto o un prestito dai 4 bit meno significativi dell'accumulatore.

continua

Tabella Flag di stato (continua)

Flag	bit n°	Funzione
P/V	2	Flag multifunzione; se ad 1: - dopo istruzioni aritmetiche indica "fuori scala" (overflow) dell'operazione - dopo istruzioni logiche o di INPUT indica che il numero di bit a 1 nell'accumulatore è dispari (Parità). Nelle istruzioni sui blocchi (Esercizio 7.1), indica che il contenuto della coppia di registri BC non è 0.
N	1	Indica quale è stata l'ultima operazione aritmetica; va ad 1 dopo una sottrazione, a 0 dopo una addizione.
C	0	Riporto o prestito (Carry / Borrow). Se ad 1 indica che c'è stato un riporto oltre il bit 7 dell'accumulatore (o un prestito per le sottrazioni).

Se ora riportiamo l'esercizio 3-1 con diversi valori di xx e yy, è possibile verificare l'effetto delle operazioni anche sui flag di stato (occorre chiudere il programma con un breakpoint e leggere la cella 0BCC).

Dati (1) **(da inserire nel programma)**		**Risultato (per operazione di ADD)**	**Flag significativi** **Bit n°**
xx	yy	(Contenuto di A)	7 6 5 4 3 2 1 0 S Z X H X P N C
00	00	0 0 (zero, positivo senza riporto)	0 1 0 0 0 0 0 0
00	01	0 1 (non zero, positivo, senza riporto)	0 0 0 0 0 1 0 0
10	70	8 0 (non zero, negativo senza riporto)	1 0 0 0 0 1 0 0
FF	01	0 0 (zero, positivo, con riporto)	0 1 0 1 0 0 0 1
EF	01	F 0 (non zero, negativo, con riporto intermedio) e così via.	1 0 0 1 0 0 1 0

(1) operandi e risultato sono rappresentanti in codice complemento a due. Il bit di Segno (7) ripete quindi il bit 7 dell'accumulatore

Il bit di stato N va ad 1 solo dopo una operazione di sottrazione.

Esercizi consigliati

- Ripetere l'esercizio 3.1 con altre operazioni (SUB, OR, AND, XOR) verificando il risultato e la parola di stato, anche sui bit H (riporto intermedio) e P (parità).

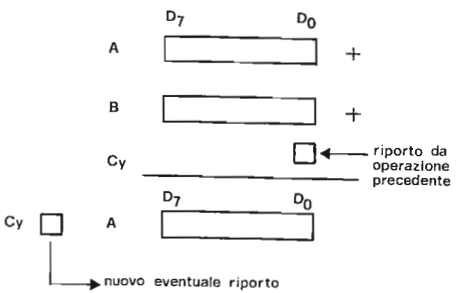
Esercizio 3.3 - Operazioni con riporto

Le operazioni aritmetiche possono essere eseguite tenendo conto del bit di riporto (Carry), proveniente da operazioni precedenti.

Le istruzioni da usare sono in questo caso:

ADC	somma con riporto	(Carry)
SBC	sottrazione con prestito	(Borrow)

Ad esempio l'operazione ADC A,B esegue:



Il bit di riporto permette di concatenare operazioni aritmetiche su parole più lunghe di 16 bit.

Possiamo ripetere l'esercizio 3.1 usando le operazioni con riporto, con l'accortezza di inserire una istruzione che fissa ad un valore determinato il bit Cy.

Questa istruzione è la SCF (Set Carry Flag):

800	37	E 3.3	SCF	Carry = 1
801	3E, xx		LD A, "xx"	Carica xx in A
803	CE, yy		ADC "yy"	Gli somma, con riporto, yy
805	76		HALT	

Ad esempio, se xx = 01, yy = 02 il risultato (registro A, cioè cella BCD) è 04 (01 + 02 + 1 = 04).

Esercizi consigliati

- Ripetere l'esercizio 3.3 usando l'istruzione "SBC"; verificare il risultato in A e sui bit di stato.

- Scrivere e verificare un programma che esegua somme tra dati di 16 bit (coppia di registri) e verificare l'esecuzione.

- Scrivere e verificare un programma che esegua la somma di dati di 24 bit:
  (traccia: dato x nei registri C,D,E
  dato y nei registri B,H,L
  Fare eseguire: HL = HL + DE,
  A = B + C + Cy
  Il risultato è in AHL).

- Scrivere, con la tecnica descritta in precedenza, un programma che esegua somme tra dati di 32 bit, prelevati da una zona di memoria.

L'istruzione "Compare"

Per determinare se due dati sono eguali usando le istruzioni note fino a questo punto possiamo, ad esempio, sottrarre il primo dal secondo e verificare se il risultato è 0. Questo però distrugge il dato contenuto precedentemente nell'accumulatore. Per queste operazioni di confronto è disponibile una istruzione apposita: la "Compare".

CP <sorgente>
↳ può essere un valore immediato, un registro o il contenuto di una cella di memoria puntata dai registri HL. Questa istruzione esegue l'operazione A - sorgente, senza alterare il contenuto di A ma solo modificando, in base al risultato, i flag di stato.
Si ha quindi:

S	= 1	Se A < sorgente
Z	= 1	Se A = sorgente
C	= 1	Se c'è stato prestito
P/V	= 1	Se c'è stato fuoriscala

In pratica, per decidere il risultato del confronto, basta osservare i bit S e Z.

Esercizio 3.4: Istruzione "CP"

Permette di verificare l'effetto della istruzione di compare sul registro di stato.

800	3E, xx	E 3.4	LD A, "xx"	primo valore
	FE, yy		CP "yy"	secondo valore
	DF		RST 18	breakpoint per verificare il risultato

Sostituire a xx e yy valori opportuni per verificare i Flag nelle diverse condizioni:

xx = yy
xx > yy
xx < yy

Verificare che il contenuto di A non è alterato.

Esercizio consigliato

- Ripetere l'esercizio 3.4 con i vari modi di indirizzamento della istruzione Compare (CP <registro>, CP <cella puntata da HL>...)

Comandi al registro di stato

Il registro di stato dello Z80 viene modificato solo dalle istruzioni logiche e aritmetiche. Un semplice trasferimento non altera i flag; ad esempio LDA,0 non mette ad 1 il flag Z.

Per far si che il registro di stato indichi effettivamente il contenuto dell'accumulatore (quindi Z = 1 se A = 0 ecc.) occorre inserire istruzioni quali AND A (AND dell'accumulatore con se stesso) oppure OR A, che settano i flag di stato ma non alterano il contenuto dell'accumulatore.

Con tecnica analoga è possibile usare istruzioni fittizie per fissare in determinate posizioni i bit di stato, senza modificare l'accumulatore. Ad esempio:

AND A, OR A	mettono C ed N a 0, H ad 1
CP A	mette Z ad 1 (con $A \neq 0$)

Le uniche istruzioni che agiscono direttamente sul registro di stato sono:

SCF:	Set Carry Flag; mette Cy = 1
CCF:	Complement Carry Flag; complemento Cy

Esercizi consigliati

Verificare quanto indicato in precedenza mediante brevi programmi del tipo:

- istruzione di trasferimento (Load);
a) - istruzione logica/aritmetica;
b) - HALT.

Ponendo un breakpoint in a) è possibile verificare, leggendo il registro di stato, che l'istruzione di Load non agisce sui Flag.

Con un breakpoint in b) si può verificare l'azione dell'istruzione logico-aritmetica sul registro di stato.

Tabella II - Istruzioni logico - aritmetiche

a) Registri di 8 bit.

	B	C	D	E	A	L	(HL)	A	n	← operando
ADD	80	81	82	83	84	85	86	87	C6,n	
ADC	88	89	8A	8B	8C	8D	8E	8F	CE,n	
SUB	90	91	92	93	94	95	96	97	D6,n	
SBC	98	99	9A	9B	9C	9D	9E	9F	DE,n	
AMD	A0	A1	A2	A3	A4	A5	A6	A7	E6,n	
XOR	A8	A9	AA	AB	AC	AD	AE	AF	EE,n	
OR	B0	B1	B2	B3	B4	B5	B6	B7	F6,n	
CP	88	89	BA	BB	BC	BD	BE	BF	FE,n	
INC	04	0C	14	1C	24	2C	34	3C		
DEC	05	0D	15	1D	25	2D	35	3D		

risultato in A

b) Registri di 16 bit.

	BC	DE	HL	SP	← operando
ADD	09	19	25	39	
ADC	ED4A	ED5A	ED6A	ED7A	
SBC	ED42	ED52	ED62	ED72	
INC	03	13	23	33	
DEC	0B	1B	2B	3B	

risultato in HL

c) Comandi al registro di stato

CCF	3F
SCF	37

4. Istruzioni di rotazione e scalamento

Una classe di istruzioni logiche permette di spostare bit a bit il contenuto dei registri a destra o a sinistra, passando o meno attraverso il Carry.

Ad esempio, se l'accumulatore contiene $B8_H$ ed il Carry è a 1, dopo l'istruzione RR A (rotate right) si ha A = DC_H e Cy = 0 (Fig. 2.8)

Istruzioni di questo tipo si possono usare per formattare un dato nel modo voluto, spostando la posizione di ciascun bit (vedi per esempio la gestione della tastiera e del visualizzatore in Appendice C); per eseguire moltiplicazioni per potenze di 2 (scalamenti

a sinistra), per far comparire in successione su una stessa posizione tutti i bit di un dato (vedi ad esempio la serializzazione software usata per gestire le cassette magnetiche in Appendice G).

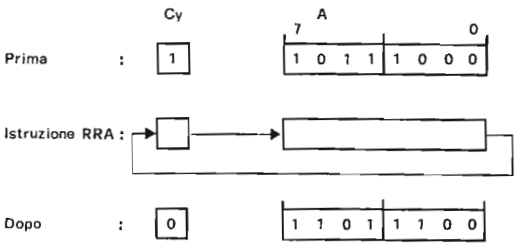
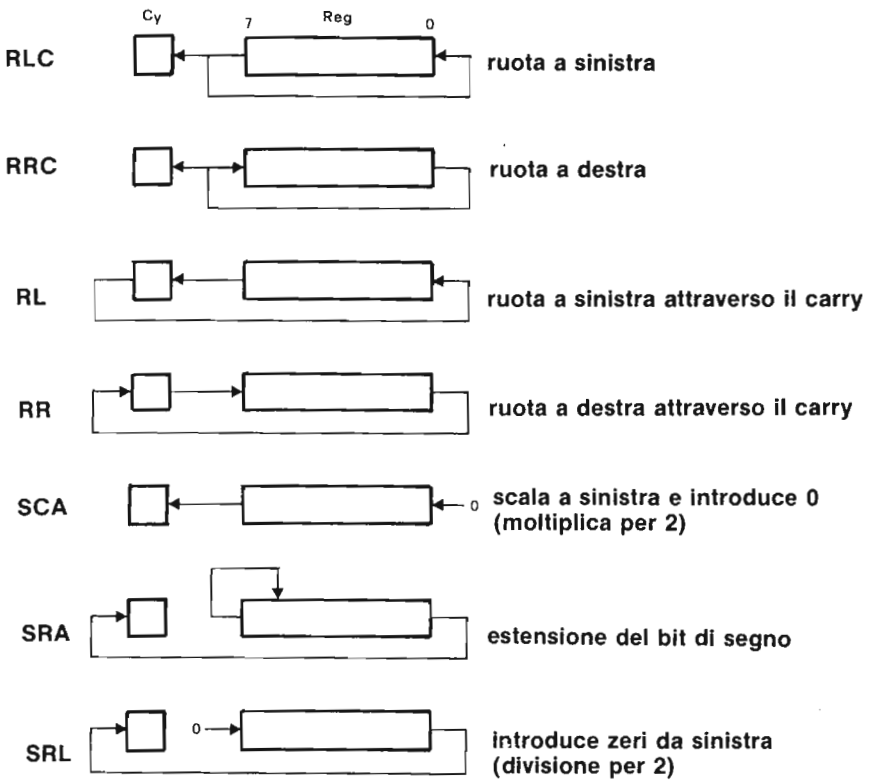


Figura 2.8 - Istruzione Rotate right.

I tipi di scalamento ammessi per lo Z80 sono:



Nello Z80 queste operazioni possono essere effettuate su qualunque registro e sulla cella di memoria puntata da HL. L'istruzione completa è:

<codice istruzione> <registro>

Se il registro è l'accumulatore A, per i primi quattro casi (RLC, RRC, RL, RR) esistono delle istruzioni speciali (riportate in tabella), formate da un solo byte e di esecuzione piú rapida.

Esercizi

-Verificare l'effetto delle istruzioni di scalamento sull'accumulatore e sul Carry:

800	37	E 4.1	SCF	Carry a 1
801	3E,xx		LDA, xx	carica in A xx
803	- -		<istr>	istruzioni di scalamento
,	DF		RST 18_H	breakpoint

Inserire, nella cella 803, le varie istruzioni di scalamento e verificare, dopo il breakpoint il contenuto di A ed il Carry.

- Verificare l'effetto delle istruzioni di scalamento su celle di memoria.

800	21,10,08	E 4.2	LD HL, yyyy	Puntatore alla cella 810_H
803	36,xx		LD(HL), xx	inizializza la cella
805	37		CF	carry a 1
806	.,..		<istr>	istruzione di scalamento
808	C9		RET	ritorno al monitor

L'effetto della istruzione di scalamento si può verificare direttamente in memoria leggendo la cella 810; per questo motivo non è stato introdotto un breakpoint ma solo un ritorno al monitor.

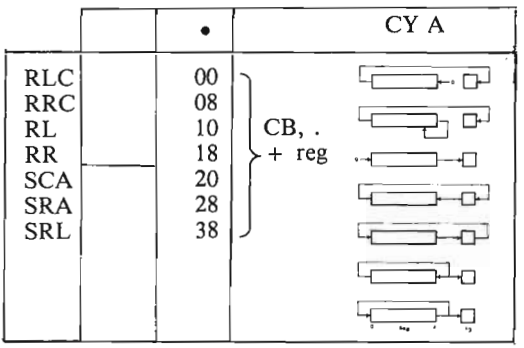
Per leggere il carry occorre inserire una RST 18_H al posto del RET.

Tabella III - Istruzioni di scalamento

a) Sull'accumulatore.

RLC	07
RRC	0F
RL	17
RR	1F

b) Su tutti i registri e su (HL).



reg.	B	C	D	E	H	L	(HL)	A
	0	1	2	3	4	5	6	7

Ad esempio:

SCA L ⇒ CB, 26

↓

20 + 6

/ \

(Codice SCA) (Codice registro L)

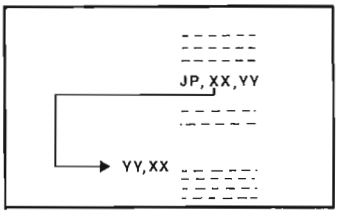
5. Istruzioni di salto e salto condizionato

Le istruzioni di salto cambiano il contenuto del program counter e provocano quindi una interruzione nella sequenzialità del programma.

Lo Z80 ha due tipi di istruzini di salto:

JP, xx, yy (3 byte): salto con indirizzo assoluto; l'istruzione successiva viene prelevata dalla cella yy xx e l'esecuzione procede da questo punto.

Esempio:

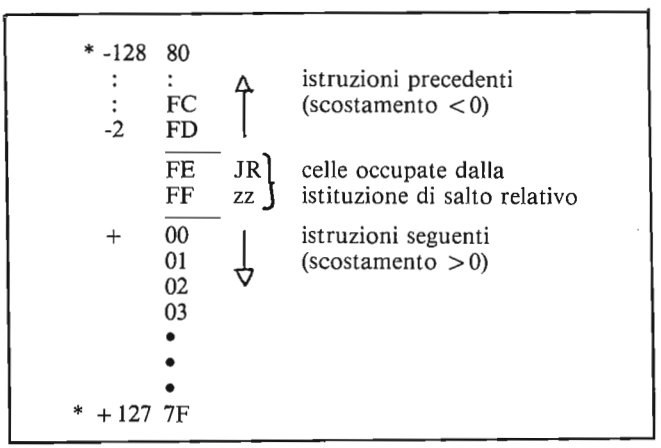


il controllo passa alla istruzione contenuta nella cella YY, XX

L'istruzione di salto assoluto permette di trasferire il controllo ad un indirizzo qualsiasi compreso nei 65K della memoria.

JR, zz (2 byte): salto con indirizzo relativo; la nuova istruzione viene prelevata dalla cella il cui indirizzo si ricava sommando zz (in complemento a 2) all'indirizzo successivo alla istruzione di salto relativo.

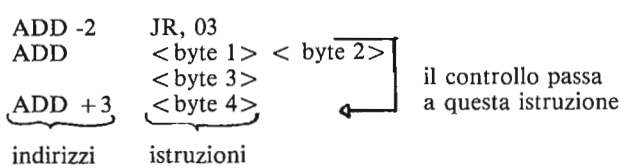
La "regola" per calcolare lo spostamento necessario per saltare in una determinata posizione è indicata in figura 2.9



* valori limite

Figura 2.9 - Calcolo dello scostamento

Ad esempio



L'istruzione JR permette di saltare solo entro un campo di + 129, -127 celle; consente anche di scrivere programmi che possono essere eseguiti in qualunque posizione dalla memoria (detti per questo "position independent").

Si consiglia comunque di usare inizialmente i salti assoluti, per evitare il calcolo (in esadecimale) dello scostamento.

L'esecuzione del salto, sia assoluto che relativo, può essere condizionata dal valore di alcuni bit del registro di stato. (Zero, parità, riporto, segno), e questo corrisponde ad una diramazione nel flusso del programma (Fig. 2.10)

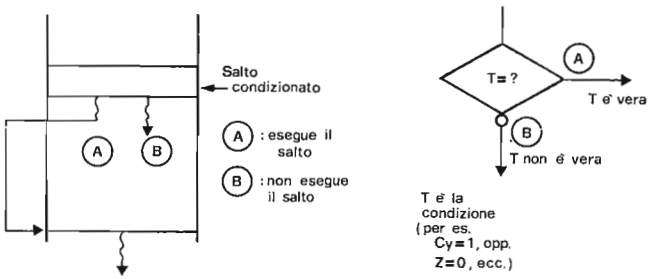


Figura 2.10 - Salto condizionato.

Ad esempio, l'istruzione JP, Z, xxxx, determina un salto all'indirizzo xxxx solo se nel registro di salto il bit Z è 1 (cioè se una precedente istruzione logica o aritmetica ha portato il contenuto dell'accumulatore a 0). Se a $A \neq 0$ l'esecuzione continua con l'istruzione immediatamente successiva a JP, Z, xxxx.

In questo caso la diramazione si indica come in fig. 2.11.

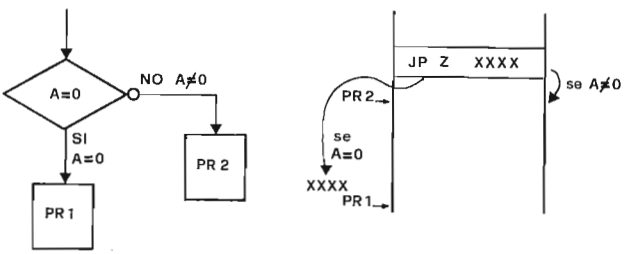


Figura 2.11 - Esempio di programma con salto condizionato dal Flag Z.

Esercizio 5.1

Questo programma usa l'istruzione di salto condizionato; per evidenziare la diramazione percorsa durante l'esecuzione vengono visualizzate sul display due cifre differenti.

Il diagramma di flusso è in fig. 2.12

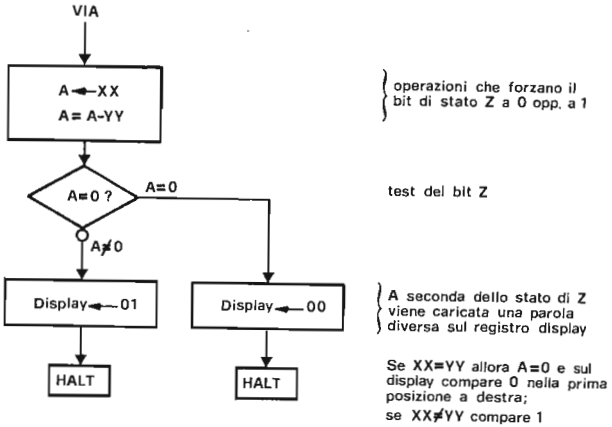


Figura 2-12

800	3E, xx	E 5-1	LD A, xx	
802	D6, yy		SUB yy	A = A - yy
804	CA,10,08		JZ DISP 0	se A = 0 (cioè xx = yy) va a DISP 0 (DISP 0 = 810)
807	3E,01	DISP 1	LD A, 01	se A ≠ 0 esegue DISP 1
809	32,00,0C		LD (0C00), A	
80/C	76		HALT	
	•			celle non utilizzate dal programma
	•			
	•			
810	3E,00	DISP 0	LD A, 00	
812	32,00,0C		LD (0C00), A	
815	76		HALT	

Esercizio 5.2

L'istruzione di salto consente di far eseguire ripetutamente lo stesso spezzone di programma; questa struttura (Fig. 2.13) prende il nome di "anello" (loop).

Possiamo sfruttare il meccanismo del loop per far comparire piú cifre sul display. Per ciascuna cifra occorre eseguire un trasferimento sul registro display (indirizzo 0C00); se questi trasferimenti sono eseguiti ripetitivamente, i segmenti attivati appaiono sempre illuminati.

Conviene usare HL come puntatore al registro CONADD (cella 0C00, vedi esercizio 2.2), per poter eseguire ciascun trasferimento con una sola istruzione (LD (HL), XX).

800	22,00,0C	E 5.2	LD HL, 0C00	Carica il puntatore
803	$36,0x_0$	LOOP	LD (HL), $0x_0$	1 cifra a destra,
805	$36,1x_1$		LD (HL), $1x_1$	trasferimento sul display
807	$36,2x_2$		LD (HL), $2x_2$	
	•		•	gruppo di trasferimenti (8 cicli)
	•		•	
	•		•	xj è una cifra esadecimale
80F	$36,6x_6$		LD (HL), 6x6	
811	$36,7x_7$		LD (HL), $7x_7$	
813	C3,03,08		JP LOOP	salto che richiude l'anello

Sul display comparirà $x_7\,x_6\,x_5\,x_4\,x_3\,x_2\,x_1\,x_0$. È naturalmente possibile far comparire piú o meno cifre cambiando il numero di LD (HL), yy nel loop.

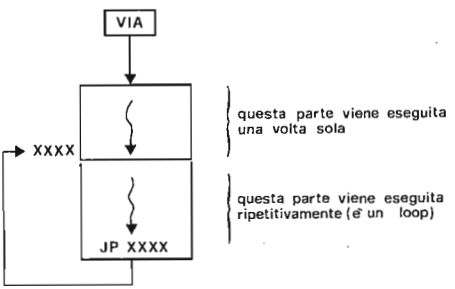


Figura 2.13 - "Anello" di programma

Esercizi consigliati

- Realizzare un programma che faccia comparire la stessa cifra in tutte le posizioni del display (traccia: - per passare da una cifra all'altra sommare 10_H ad A come per l'esercizio 5.1).
- Ripetere l'esercizio 5.1 sfruttando altri bit di condizione (riporto, parità, segno).
- Ripetere gli esercizi 5.1 e 5.2 usando le istruzioni di salto relativo. Provare a scrivere questi programmi in altre zone di memoria RAM (per esempio da 880_H, o da 900_H ecc.) e verificare che il funzionamento è indipendente dalla posizione.

Tabella IV - Istruzioni di salto

JP, <add L>, <add H>	JR disp	Condizione
C3, n, n	18, n	—
C2, n, n	20, n	Z = 0 (A≠0)
CA, n, n	28, n	Z = 1 (A = 0)
D2, n, n	30, n	Cy = 0 (non riporto)
DA, n, n	38, n	Cy = 1 (riporto)
E2, n, n	-	P = 0 (parità pari)
EA, n, n	- non disponibili	P = 1 (parità dispari)
F2, n, n	- come JR	S = 0 (A≥0)
FA, n, n	-	S = 1 (A<0)

Tabella V - Scostamenti per istruzioni di salto relativo all'indietro.

DEC	HEX	JR CODE	DEC	HEX	JR CODE	DEC	HEX	JR CODE	DEC	HEX	JR CODE
1	01	FD	33	21	DD	65	41	BD	97	61	9D
2	02	FC	34	22	DC	66	42	BC	98	62	9C
3	03	FB	35	23	DB	67	43	BB	99	63	9B
4	04	FA	36	24	DA	68	44	BA	100	64	9A
5	05	F9	37	25	D9	69	45	B9	101	65	99
6	06	F8	38	26	D8	70	46	B8	102	66	98
7	07	F7	39	27	D7	71	47	B7	103	67	97
8	08	F6	40	28	D6	72	48	B6	104	68	96
9	09	F5	41	29	D5	73	49	B5	105	69	95
10	0A	F4	42	2A	D4	74	4A	B4	106	6A	94
11	0B	F3	43	2B	D3	75	4B	B3	107	6B	93
12	0C	F2	44	2C	D2	76	4C	B2	108	6C	92
13	0D	F1	45	2D	D1	77	4D	B1	109	6D	91
14	0E	F0	46	2E	D0	78	4E	B0	110	6E	90
15	0F	EF	47	2F	CF	79	4F	AF	111	6F	8F
16	10	EE	48	30	CE	80	50	AE	112	70	8E
17	11	ED	49	31	CD	81	51	AD	113	71	8D
18	12	EC	50	32	CC	82	52	AC	114	72	8C
19	13	EB	51	33	CB	83	53	AB	115	73	8B
20	14	EA	52	34	CA	84	54	AA	116	74	8A
21	15	E9	53	35	C9	85	55	A9	117	75	89
22	16	E8	54	36	C8	86	56	A8	118	76	88
23	17	E7	55	37	C7	87	57	A7	119	77	87
24	18	E6	56	38	C6	88	58	A6	120	78	86
25	19	E5	57	39	C5	89	59	A5	121	79	85
26	1A	E4	58	3A	C4	90	5A	A4	122	7A	84
27	1B	E3	59	3B	C3	91	5B	A3	123	7B	83
28	1C	E2	60	3C	C2	92	5C	A2	124	7C	82
29	1D	E1	61	3D	C1	93	5D	A1	125	7D	81
30	1E	E0	62	3E	C0	94	5E	A0	126	7E	80
31	1F	DF	63	3F	BF	95	5F	9F			
32	20	DE	64	40	BE	96	60	9E			

Questa tabella è comoda per assemblare "a mano". Fornisce lo scostamento da usare per istruzioni di salto relativo all'indietro, partendo dal numero di byte da saltare, in decimale o in esadecimale.

I salti in avanti si calcolano facilmente per via diretta, come indicato a pag. 34.

Ad esempio, nel programma:

```
LOOP    LD      A,xx
        LD      B,A
        JR      LOOP        (-3 bytes)
```

Il displaciment per la istruzione di salto relativo è FB_H, come si ricava dalla tabella in corrispondenza al valore 3 (salto di 3 byte indietro).

Esercizio 5.3

Il programma proposto in questo esercizio combina ed utilizza tutti gli esempi di istruzioni visti fino a questo punto. Scopo dell'esercizio è far comparire 2 cifre diverse sul display a seconda dello stato di una linea di interfaccia (registro di ingresso).

Possiamo per questo sfruttare uno qualsiasi dei bit non utilizzati per la gestione della tastiera (il visualizzatore non usa linee di ingresso). Si sceglie, a titolo di esempio, il bit 5, corrispondente al piedino 6 del connettore B sulla scheda di CPU (vedi fig. 1.31 pag. 10), come indicato in figura 2.14.

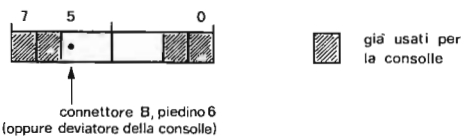


Figura 2.14 - Assegnazioni registro di ingresso.

Il circuito da montare per poter verificare questo esercizio è in figura 2.15.

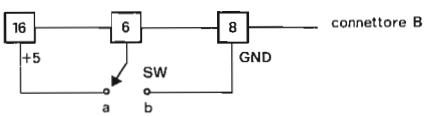


Figura 2.15 - Cablaggio esterno.

Lo stato dei vari bit del registro di ingresso può essere osservato direttamente sul display, aprendo la cella $0C00_H$. In queste condizioni sul campo dati compare direttamente il contenuto del registro. È in pratica possibile osservare solo i bit da 2 a 6, perchè 0,1

e 7 sono occupati per la gestione della tastiera e quando si preme un tasto il visualizzatore è bloccato.

Forzando a massa i piedini 3,4,5,6,8 del connettore B si vedrà cambiare il campo dati. Le linee lasciate libere vanno al livello 1 logico.

Il programma deve:

- Leggere lo stato del deviatore (istruzione di trasferimento).
- Isolare il bit 5 (operazione logica).
- Eseguire due tratti di programma diversi a seconda dello stato del deviatore (salto condizionato).
- I due sottoprogrammi devono determinare la visualizzazione di due cifre diverse (vedi es. 5.2)

Diagramma di flusso

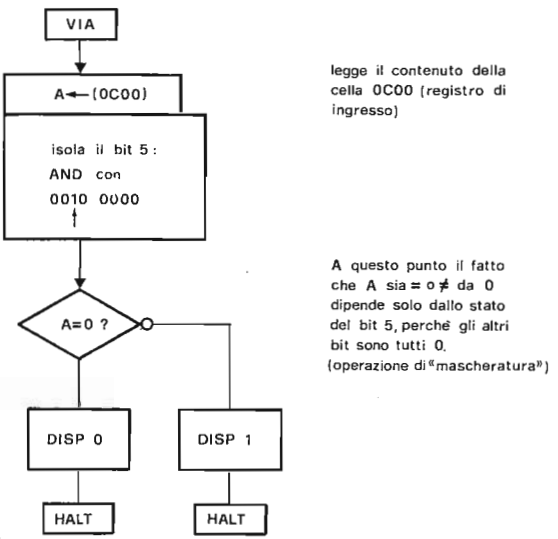


Figura 2.16 - Programma di test di una linea di ingresso.

Per far eseguire correttamente il programma occorre prima disporre il deviatore SW nella posizione voluta e quindi dare il comando GOTO.

Esercizio consigliato

- Ripetere il programma E - 5.3 chiudendo i due spezzoni DISP0 e DISP 1, anzichè con HALT, con un salto a READ; in questo modo l'esecuzione è continuamente ripetuta ed il deviatore può essere azionato piú volte, dopo aver lanciato il programma.

800	21,00,0C	E 5-3	LD HL, 0C00	Prepara il puntatore
803	7E	READ	LD A, (HL)	legge il registro 0C00
804	E6,20		AND 20	Operazione di mascheratura del bit 5 : 20_H è 0010 0000 ↑ bit 5 / \ =1 =0 A≠0 A=0 Questa operazione predispone il flag Z a 0 o a 1
805	C2,10,08		JNZ, DISP 1	se Z = 0 (A # 0), va a DISP 1
808	3E,00	DISP 0	LD A, 00	se A = 0 esegue DIP 0
80A	77		LD (HL), A	
80B	76		HALT	
810	3E,01	DISP 1	LD A, 01	A≠0; esegue DISP 1
812	77		LD (HL), A	
813	76		HALT	

Esercizio 5.4 - Ritardi

L'esecuzione ripetuta di una sequenza di istruzioni può essere utilizzata per inserire ritardi in un programma. Occorre per questo usare un contatore (registro o cella di memoria), predisposto ad un determinato valore, che viene decrementato di una unità ogni volta che viene percorso il loop. Quando il contatore raggiunge il valore 0 il programma esce dal loop e termina il ritardo.

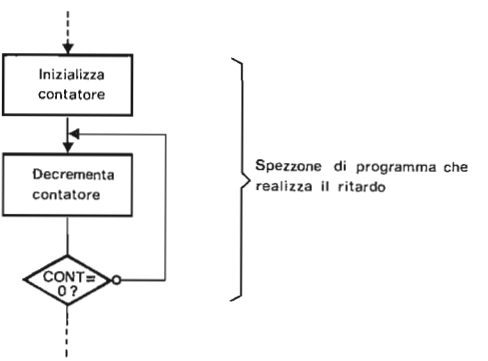


Figura 2.17 - Esempio di ritardo ottenuto dal programma.

Vediamo ora un esempio di inserimento di ritardo in un programma.

800	21,00,0C	E 5.4	LD HL, 0C00	: puntatore display
803	71	LOOP	LD (HL), C	: trasferisce C sul display
804	0C		INC C	: incrementa N
805	00		NOP	}
806	00		NOP	} Spazio per inserire il ritardo
807	00		NOP	}
808	C3,03,08	END	JP LOOP	chiude l'anello: LOOP = 803

Questo programma fa comparire tutte le cifre esadecimali 0.....F successivamente in tutte le posizioni del display. Infatti il registro C assume in sequenza i valori in tabella:

```
00 ┐          ┌0
01 │          │1
02 │ cifre    │2   in posizione 0
 • │          │•
 • │          │•
 • │          │.
0F ┘          └F
10 ┐          ┌0
11 │ cifre    │1   in posizione 1   Vedi Figura 1.10
 • │          │•
 • │          │•
1F ┘          └F
20 ┐          ┌•
 • │          │•
 • │          │•
 • │          │•
FD ┐          ┌D
FE │ cifre    │E   in posizione 7
FF ┘          └F
00
 •
 •
```

Eseguendo direttamente il programma E 5.2 la cadenza di ripetizioni è molto alta, tale che tutti i segmenti di tutte le cifre appaiono contemporaneamente accesi. Per visualizzare l'effettiva successione di operazioni occorre inserire nell'anello un ritardo: per questo sostituiamo ai tre NOP un salto al programma di ritardo DELAY: JP DELAY.

...	...	...	
805	C3,10,08	JP, DELAY	Salto al sottoprogramma
...	...	...	di ritardo DELAY = 810

- Scriviamo DELAY a partire dalla cella 810.

810	3E, FF	DELAY	LDA, FF	Inzializza il contatore A
802	3D	DELOP	DEC A	lo decrementa
803	C2,12,08		JP NZ DELOP	se A≠0 ripete DELOP=812
806	C3,08,08		JP END	ritorna al programma E 5.2

Possiamo verificare che il programma DELAY inserisce effettivamente un ritardo, in quanto dopo la modifica è possibile seguire lo spostarsi delle cifre da una posizione all'altra su display. L'entità del ritardo dipende dal valore caricato inizialmente in A.

Esercizi consigliati

- Ripetere l'esercizio 5.4 con diversi valori di ritardo.
- Determinare quantitativamente il ritardo introdotto caricando il contatore con FF (traccia: conteggiare il numero dei cicli richiesti per l'esecuzione di ogni istruzione).

È possibile fare una verifica sperimentale di questo risultato, ad esempio cronometrando il tempo richiesto per ciclare un numero definito di volte tutte le cifre sul visualizzatore.

Esercizio 5.5 - Istruzione DJNZ

Lo Z80 possiede una istruzione speciale per implementare la struttura di controllo di programmi da eseguire in modo ripetitivo quali ad esempio gli anelli di ritardo degli esercizi precedenti. Si tratta della istruzione DJNZ, <xx> (Decrement Jump Not Zero), che esegue le seguenti operazioni:

- decrementa il contenuto del registro B, che viene usato come contatore del numero di cicli dell'anello di programma;

- verifica il contenuto di B; se diverso da 0 esegue un salto relativo con scostamento xx; se B = 0 continua l'esecuzione con l'istruzione successiva.

Il codice della DJNZ è: 10, <scostamento>.

Lo scostamento si determina con le stesse regole già indicate per le istruzioni di salto relativo.

Il modulo "DELAY" dell'esercizio 5.4 diventa, utilizzando questa istruzione:

810	06, FF	DELAY	LD B, FF_H	; inizializza B come contatore
802	10, FE	DELOP	DJNZ, DELOP	; decrementa B, se $\neq 0$ salta a DELOP
804	C3,08,08		JP END	; ritorna nel programma principale

6. Chiamata di sottoprogrammi

Un'altra classe di istruzioni di salto permette di riprendere l'esecuzione, dopo un determinato comando, dalla istruzione successiva a quella di partenza del salto. Queste istruzioni di salto sono i "CALL" (chiamata a subroutine); il comando è l'istruzione di RET (Return), che deve chiudere il sottoprogramma chiamato.

Il meccanismo dei CALL e RET è indicato in figura 2.18.

Si noti in particolare che uno stesso sottoprogramma può essere chiamato, con il meccanismo CALL/RET da piú parti del programma principale, rientrando sempre poi nel punto corretto, cioè all'istruzione successiva al CALL.

Questo avviene perchè l'indirizzo di quest'ultima, cioè l'indirizzo di ritorno, viene "salvato" dalla istruzione CALL stessa in una particolare zona di memoria, detta "Stack", dalla quale viene ripreso e trasferito nel program counter della CPU all'esecuzione del RET.

Il modo di funzionamento dello Stack verrà spiegato piú in dettaglio in seguito.

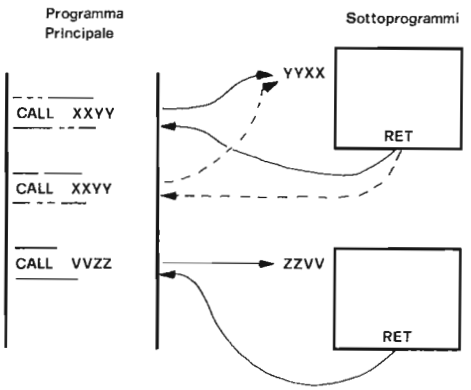


Figura 2.18 - Isruzioni di CALL e Return.

Esercizio 6.1

Gli esercizi 5.5 e 5.4 possono essere scritti utilizzando una CALL per il modulo di ritardo operando come segue:

- Sostituire i tre NOP con CALL DELAY.

...	...		...
805	CD,10,08		CALL DELAY
...	...		...

- Chiudere il programma DELAY con RET

..	..		..
814	C9		RET

Nel programma di Monitor residente su ROM esiste già un programma di ritardo (utilizzato per eliminare i rimbalzi dei contatti della tastiera e per fissare la cadenza dei dati nei programmi di registrazione e rilettura da cassette). Questi programmi comprendono un modulo base RATE che produce un ritardo di 1,6 ms circa, ed i moduli RATE P2, -P4, - 08 e -D2, -D4, che rispettivamente moltiplicano per 2, 4, 8 e dividono per 2, 4 il ritardo base.

La gamma di ritardi e gli indirizzi dei sottoprogrammi sono riportati in tabella.

Programma di ritardo	Ritardo (ms)	Indirizzo
RATE P8	13,33	01AF
RATE P4	6,66	01B2
RATE P2	3,33	01B5
RATE	1,66	01B8
RATE D2	0,833	01BB
RATE D4	0,416	01BE

Tutti questi programmi usano come contatore il registro A.
Sostituendo nell'esercizio 5.4 una chiamata ai vari sottoprogrammi di ritardo al posto dei NOP agli indirizzi 805/6/7, si può verificarne l'effetto sul movimento delle cifre del visualizzatore, e cronometrare i tempi ottenuti.

Esercizio 6.2 - Ritardi lunghi

Per inserire un ritardo maggiore di quello ottenibile con un solo registro di 8 bit esistono due alternative:

- Usare coppie (o n-ple) di registri
- Concatenare piú anelli di programma

In quest'ultimo caso si ha una serie di chiamate successive e si parla di "nidificazione" (nesting) dei sottoprogrammi (Fig. 2.19).

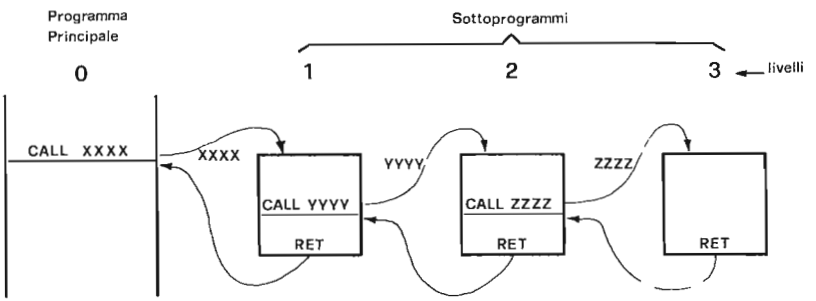


Figura 2.19 - Sottoprogrammi nidificati.

Nel nostro caso possiamo costruire un primo livello di loop che chiama piú volte RATE (usato come sottoprogramma di secondo livello):

- Inserire (sempre al posto dei NOP) un CALL DELOOP nel programma E 5.4

..	...		....
805	CD,10,08		CALL DELOOP
..	...		....

- Programma DELOOP.

810	06,FF	DELOOP	LD B, FF	
812	CD,B8,01,	RIT	CALL RATE	viene chiamata 255 volte (FF hex)
815	10,FB		DJNZ RIT	se B≠0 torna a RIT
817	C9		RET	

Si noti l'uso della istruzione DJNZ, YY (Decrementa B e salta se ≠ 0) per realizzare il contatore dell'anello.

Esercizi consigliati

- Realizzare lo stesso ritardo con uno o due anelli nidificati, variando il parametro di inizializzazione del contatore.
  Attenzione: Bisogna usare registri diversi per i due anelli.
- Determinare i ritardi minimo e massimo per DELOOP - Come deve essere inizializzato B?

Contatori con coppie di registri

Per ripetere un anello piú di 255 volte possiamo utilizzare come contatore una coppia di registri. Occorre in questo caso verificare quando entrambi i registri della coppia sono a zero, per uscire dall'anello. Ricordiamo che le istruzioni logico - aritmetiche (in particolare INCrement e DECrement) su coppie di registri non modificano i bit del registro di stato.

Una sequenza che verifica se il contenuto dei registri B e C è uguale a 0 è, per esempio:

```
LD A, B
OR C
```

Al termine della seconda istruzione A è zero solo se tanto B che C erano a zero. L'ultima istruzione (OR C) predispone sullo stato corretto il Flag di zero.

Esercizi consigliati

- Realizzare un ritardo lungo usando una coppia di registri.
- Realizzare un ritardo pari a 30'' concatenando anelli ove il contatore è una coppia di registri. Visualizzare i cicli di ritardo incrementando, ogni 30'' una cifra sul display.

Istruzioni di chiamata e ritorno condizionate

Analogamente a quanto visto per le istruzioni di salto relativo ed assoluto, anche per le chiamate di sottoprogrammi (CALL) ed i ritorni (RET) è possibile condizionare la esecuzione della istruzione mediante il contenuto del registro di stato.

Se la condizione non è verificata viene eseguita l'istruzione successiva, senza salto o ritorno.

Esercizi sulle chiamate e ritorni condizionati sono nel paragrafo 6 (pag. 43).

Istruzioni di RESTART

Lo Z80 prevede, analogamente al precedente 8080, un gruppo di istruzioni di chiamata (CALL) a sottoprogrammi in posizioni fisse di memoria. Queste sono le Restart (RST).

Esistono 8 Restart che equivalgono a tutti gli effetti a dei CALL con indirizzo di salto implicito; esse richiamano l'esecuzione dei programmi che iniziano alle celle 00, 08_H, 10_H.... 38_H.

Queste istruzioni sono codificate su un unico byte e vengono usate anche nella gestione delle interruzioni.

Tabella VI - Istruzioni di chiamata e ritorno condizionate

CALL,n,n	RET	Condizione	
CD,n,n	C9	—	
C4,n,n	C0	Z=0	(A ≠ 0)
CC,n,n	C8	Z=1	(A = 1)
D4,n,n	D0	C=0	no riporto
DCn,n	D8	C=1	riporto
E4,n,n	E0	P=0	
EC,n,n	E8	P=1	
F4,n,n	F0	S=0	(A ⩾ 0)
FC,n,n	F8	S=1	(A < 0)

Istruzioni di RESTART	
RST 00	C7
RST 01	CF
RST 10	DF
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF

Gestione ed uso dello stack

Lo "stack" (letteralmente "pila") è una zona di memoria accessibile oltre che attraverso le istruzioni di LoaD, anche mediante altre istruzioni speciali che lo gestiscono con tecnica sequenziale LIFO (Last In - First Out); l'ultimo dato immagazzinato è il primo ad essere letto.

Se, ad esempio, viene immagazzinata nello stack la sequenza: 12, 27, 68, 32, verrà riletta come 32, 68, 27, 12 ("32", caricato per ultimo, viene letto per primo).

Un registro interno dello Z80 ha funzione di puntatore alla memoria esterna usata come stack (stack pointer: SP).

Le istruzioni di trasferimento speciali da e verso lo stack sono:

PUSH <coppia di registri>: trasferisce il contenuto di una coppia di registri nello stack.

POP <coppia di registri>: legge in una coppia di registri il contenuto dello stack.

Dopo una prima inizializzazione eseguita con l'istruzione LD SP, <valore di 16 bit>, il contenuto dello SP è sempre aggiornato in modo automatico, per cui è sufficiente tener conto dell'ordine secondo cui i dati sono inseriti e prelevati dallo stack.

Lo stack viene usato per memorizzare il Program Counter nelle chiamate a sottoprogrammi (CALL) e per ripristinarlo nei ritorni (RET).

Può servire anche per "salvare" i registri che sono utilizzati dalla routine chiamata allo scopo di ripristinarne correttamente il contenuto al ritorno nel programma chiamante.

È evidente che il meccanismo dello stack opera correttamente solo se lo stack pointer punta ad una zona di RAM. Qualunque programma che fa uso dello stack, sia per il salvataggio di registri che per dei semplici CALL, deve innanzitutto inizializzare lo stack.

Questo non è stato fatto negli esercizi precedenti e non è necessario nel caso del Picocomputer perchè il programma di Monitor stesso provvede ad assegnare, all'atto del reset, un valore corretto (di "default") allo Stack Pointer di utente, attraverso il meccanismo di ripristino dei registri del comando GOTO.

Tale valore può essere modificato da istruzioni esplicite di LD SP, <nuovo valore> presenti nel programma di utente.

Per semplificare il caricamento dei codici, l'inizializzazione dello stack non compare neanche negli esercizi successivi, ma deve essere considerata come eseguita implicitamente all'atto del GOTO.

Esercizio 6.3 - Loop nidificati con salvataggio dei registri

In alternativa alla tecnica usata nell'esercizio 6.2, il ritardo lungo con due anelli concatenati può usare sempre il registro B come contatore, con l'avvertenza di salvare nello stack il valore di B nell'anello esterno (PUSH BC), prima di chiamare il secondo, e di ripristinarlo (POP BC) dopo la chiamata. La prima parte dell'esercizio 5.4 diventa quindi:

810	06,XX	DEL 1	LD B, RIT2	n° cicli anello esterno
802	C5	LOP 1	PUSH BC	salva B e C
803	CD,20,08		CALL DEL2	chiamata II ritardo che usa B
806	C1		POP BC	ripristina B e C
807	10,F9		DJNZ LOP1	se B ≠ 0 torna a LOP1
A	C9		RET	
820	06,YY	DEL 2	LD B, RIT2	n° cicli II loop
822	10,FF	LOP 2	DJNZ LOP2	se B ≠ 0 ripete
824	C9		RET	

Per salvare piú registri occorre una sequenza di PUSH:

```
PUSH AF
PUSH BC
PUSH DE
PUSH HL
  :
```

e per ripristinarli l'inversa sequenza di POP:

```
  :
POP HL
POP DE
POP BC
POP AF
  :
```

Tabella VII - Istruzioni di trasferimento sullo stack

I codici operativi per le istruzioni di PUSH e POP sono:

registri	PUSH	POP
AF	F5	F1
BC	C5	C1
DE	D5	D1
HL	E5	E1

7. Istruzioni su blocchi di dati

Lo Z80, unico in questo tra i micro ad 8 bit della sua generazione, possiede alcune istruzioni che operano su un blocco di dati contenuto nella memoria.

Le operazioni ammesse per queste istruzioni sono:

Load (LD):	trasferimento da memoria a memoria
Compare (CP):	confronto tra contenuto dell'accumulatore e celle di memoria.
Input (IN):	trasferisce in una zona di memoria i dati presenti su una porta di ingresso.
Output (OUT):	trasferisce su una porta di uscita il contenuto di una zona di memoria.

Queste istruzioni utilizzano HL come puntatore alla sorgente, DE come puntatore alla destinazione (se prevista), e BC come contatore delle operazioni da eseguire che viene automaticamente decrementato ad ogni esecuzione.

Il codice mnemonico completo è generato combinando il codice dell'operazione (LD oppure CP), con dei suffissi che indicano il verso di aggiornamento dei puntatori (Increment e Decrement) e l'eventuale ripetizione della esecuzione (Repeat). Quest'ultima ha luogo se il contatore BC è diverso da 0.

Avremo così, per esempio:

LDIR

LoaD — Repeat: l'istruzione viene ripetuta fino a quando BC = 0

Increment: HL e DE sono incrementati dopo ogni operazione

CPD

Compare — Decrement: il puntatore alla sorgente (cioè HL) viene decrementato

Quest'ultima istruzione non viene ripetuta automaticamente. E così via.

La possibilità di incrementare o decrementare i puntatori permette di trasferire tabelle di dati anche se il campo di indirizzi della zona sorgente e della destinazione sono parzialmente sovrapposti.

Le istruzioni su blocchi devono essere precedute dal caricamento dei puntatori e del contatore.

Avremo, ad esempio, per trasferire i dati da 1000_H a $10FF_H$ in 4000_H la sequenza di istruzioni:

```
LD   HL, 1000H     ; puntatore sorgente
LD   DE, 4000H     ; puntatore destinazione
LD   BC, 0FFH      ; lunghezza blocco
LDIR               ; istruzione di trasferimento del blocco (LoaD, Increment,
                   Repart).
```

L'operazione di trasferimento (HL)→(DE) viene eseguita 255 volte (FF_H), partendo da HL = 1000 e DE = 4000.

Esercizio 7.1

Si vuole ricopiare parte del programma di monitor (ad esempio da 0000 a 0200) nella RAM. Occorre per questo inizializzare i registri come indicato nell'esempio precedente. Il programma va scritto fuori dalla zona di RAM interessata al trasferimento, per non essere cancellato dalla sua stessa esecuzione.

0B00	210000	E 7.1	LH HL,0000	Indirizzo iniziale sorgente
0B03	110008		LD DE,0800	Indirizzo iniziale destinazione
0B06	010002		LD BC,0200	Lunghezza blocco
0B09	ED B0		LD 12	Trasferimento del blocco
0B0B	C9			Ritorno al monitor

L'esecuzione di questo programma può essere verificata leggendo la RAM e la ROM ad indirizzi scalati di 800_H (ad esempio 0183_H e 0983_H); il contenuto delle due celle sarà identico.

Le istruzioni su blocchi di dati IN e la OUT (con i suffissi già descritti) non sono descritte in maggior dettaglio perchè, come già detto, il Picocomputer non ha registri mappati nello spazio di indirizzamento di I/O.

Esercizi consigliati

- Scrivere il programma che, usando istruzioni su blocchi, muove:
- da 800_H a 880_H in 900_H.
- da 800_H a 880_H nella zona che inizia a 840_H (zone sovrapposte)
- da 880_H a 900_H nella zona che inizia a 840_H.
  Negli ultimi due casi occorre in particolare scegliere tra LDIR e LDDR.
- Scrivere, usando LDIR, un programma che riempie una zona di memoria con uno stesso dato contenuto inizialmente nell'accumulatore.
- Scrivere i programmi proposti nei due esercizi precedenti senza usare le istruzioni sui blocchi.

Le istruzioni di Compare su blocchi servono per cercare un dato assegnato in una tabella.

A seguito di ciascuna operazione di Compare tra A ed (HL), i flag di stato vengono settati come per la normale istruzione di Compare tra registri.

Per di piú, se l'istruzione è con ripetizione automatica (CPIR, CPDR), se A = (HL) si termina l'esecuzione.

Il flag P/V va ad 1 quando BC = 0, cioè al termine della scansione del blocco. Questo permette di verificare se l'istruzione è stata conclusa perchè A = (HL) (Flag Z = 1) o perchè BC = 0 (P/V = 1).

Esercizio 7.2

Ricerca di un dato in tabella.

Si vuole trovare la cella della zona del monitor tra 000 e 0FF che contiene il dato "47". Tale indirizzo viene presentato sul visualizzatore di consolle.

Il diagramma di flusso di un programma che esegue queste operazioni è in Fig. 2.20.

Per presentare un dato sul display, lo si carica sul registro DE prima di tornare al monitor.

800	21,00,00	E 7.2	LD HL,TABIN12	inizio tabella
803	01,FF,00		LD BC,LUNGH	lunghezza tabella
805	3E,47		LD A, 47	
807	ED B1		CPIR	istruzione CP su blocco
809	C2,10,08		JP NZ,NOMAT	se Z = 0 non c'erano dati uguali a 47
C	EB		EX HL,DE	
D	C9		RET	scambia HL con DE ritorna al monitor presentando DE su
810	11,FF,FF	NOMAT	LD DE,FFFF	ADD carica DE ritorna al moni-
803	C9		RET	tor

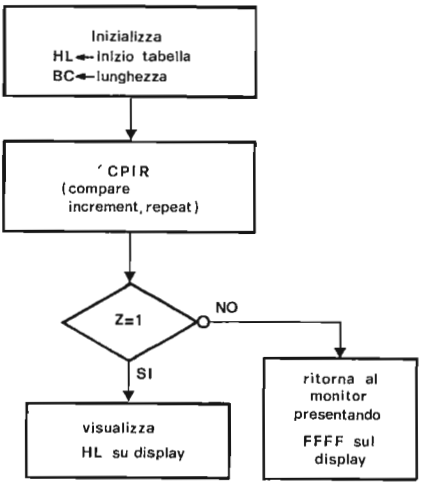


Figura 2.20 - Programma per la ricerca di un dato in una tabella.

Tornando al monitor il visualizzatore presenta il contenuto di DE sul campo ADD. Comparirà 0092, in quanto 47_H è presente all'indirizzo 0091, e l'incremento del puntatore (cioè HL), è successivo a "Compare" (quindi si esce dalla CPIR con HL = 91 + 1 = 92).

Sostituendo nella cella 804 90 ad FF e lanciando il programma, si presenterà sul display FFFF, perchè le celle da 0 a 90 non contengono 47_H.

Tabella VIII - Istruzioni su blocchi

	I	IR	D	DR			
LD	EDA0	EDB0	EDA8	EDB8	(DE)	←	(HL)
CP	EDA1	EDB1	EDA9	EDB9	A	↔	(HL)
IN	EDA2	EDB2	EDAA	EDBA	(HL)	←	
OUT	EDA3	EDB3	EDAB	EDBB	←	(HL)	

Esempio:

```
LDDR:  ED,  B8
OUTI:  ED,  A3
```

8. Istruzioni di test e comando su singoli bit

In questo paragrafo vedremo come realizzare, utilizzando opportune istruzioni, l'equivalente di circuiti monostabili o oscillatori astabili, anche con periodo variabile.

Il diagramma di flusso di un programma che realizza un astabile su una linea di uscita è in fig. 2.21.

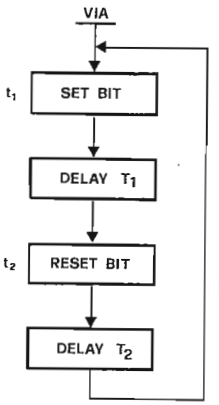


Figura 2.21

La forma d'onda rilevabile sull'uscita è in fig. 2.22
BIT è la variabile di uscita, nel nostro caso una linea del registro di uscita CONADD, ad esempio i bit 5 o 6, non utilizzati dalla consolle.

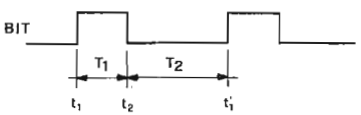


Figura 2.22

Per forzare ad un livello logico determinato uno o piú bit di un registro, esistono due possibilità:

a) utilizzare le istruzini di AND e OR con apposite "maschere". Ad esempio AND $F7_H$ (la mascheraq è F7), mette a 0 il bit 3 dell'accumulatore (unico bit a 0 della maschera); OR 10_H mette ad 1 il bit 4. (unico bit ad 1 della maschera) perchè nell'operazione OR prevale lo 1. In entrambi i casi gli altri bit non vengono alterati. Con le maschere opportune è possibile settare o resettare qualsiasi combinazione di bit sull'accumulatore.
b) Con apposite istruzioni dello Z80, (SET e RES) che consentono di forzare selettivamente ad 1 o 0 un determinato bit di qualsiasi registro. Ad esempio:

SET 5,C mette ad 1 il bit 5 del registro C
RES 1,H mette a 0 il bit 1 del registro H
SET 3,A mette a 1 bit 3 di A (come esempio in a)

e cosí via.

Esercizio 8.1 - Oscillatore astabile

Il programma che realizza le operazioni indicate in fig. 2.21 utilizzando queste ultime istruzioni è E 8.1

800	21,00,0C	E 8.1	LD HL,CONADD	puntatore al display
803	CB,EE	OSCILL	SET 5, (HL)	mette a 1 il bit 5 della cella puntata da HL
805	CD,10,08		CALL DEL 1	ritardo T_1
808	CB, EE		RES 5, (HL)	bit 5 a 0
80A	CD,20,08		CALL DEL 2	ritardo T_2
80D	C3,03,08		JP OSCILL	ripete il ciclo

DEL 1 e DEL 2 sono programmi di ritardo (vedi E 5.4), scritti a partire dalle celle 810 e 820, oppure le routine di ritardo del Monitor (vedi E 6.1).

Esercizi consigliati

- Completare E 8.1 con i due sottoprogrammi di ritardo e verificare l'esecuzione con oscilloscopio, collegato al pin opportuno dello zoccolo B.

- Modificare E 8.1 in modo che l'output sia costituito da due diverse cifre del display (traccia: sostituire alle istruzioni SET (HL) e RES (HL) dei LD (HL), XX). In questo caso è necessario inserire dei ritardi molto lunghi per verificare "a vista" l'esecuzione (Esercizio 5.4).

Esercizio 8.2 - Monostabile

Per realizzare il monostabile occore prevedere un ingresso di attivazione; questo può essere uno dei bit di CONADD non utilizzati, ad esempio il bit 3 (piedino 4 del connettore B).

Il diagramma di flusso comprenderà ora un test sull'ingresso; come indicato in fig. 2.23.

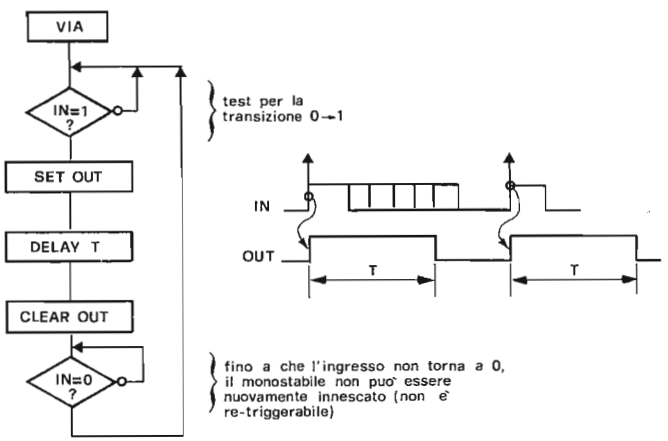


Figura 2.23

Anche per verificare lo stato di una linea di ingresso abbiamo due alternative:

a) mascherare il bit in oggetto sull'accumulatore con una operazione logica opportuna; ad esempio:

AND 10_H mette il flag Z a 1 se il bit 4 del'accumulatore è 0.

Questa tecnica è già stata usata nell'esercizio 5.3.

b) Usare l'apposita istruzione Z80 (BIT), analoga alle SET e RES: BIT 2, D mette il flag Z ad 1 se il bit 2 del registro D è a 0.

Il programma necessario per realizzare un "monostabile software" con le istruzioni BIT,SET,RES è E8.2. La linea di uscita è sempre il bit 5; la linea di ingresso è il bit 3, corrispondenti rispettivamente ai piedini 11 e 4 del connettore B.

800	21,00.0C	E8.2	LD HL,0C00	puntare al registro di I/O
803	CB,AE		RES 5,(HL)	uscita = 0
805	CB,5E	TEST1	BIT 3,(HL)	test sull'ingresso
807	28,FC		JRZ TEST1	se = 0 attende
809	CB,EE		SET 5, (HL)	se = 1 allora uscita = 1
80B	CD,..,..,		CALL DELAY	ritardo (routine fornita dall'utente o RATE del monitor.
80E	CB,AE		RES 5, (HL)	uscita = 0
810	CB,5E	TEST2	BIT3, (HL)	test in ingresso
812	20,FC		JR NZ,TEST2	se = 1 attende
814	C3,05,08		JP TEST1	se = 0 torna a cercare il fronte di salita.

Esercizi consigliati

- Usare come ingresso il bit 3 (pin 4 conn. B) connesso ad un generatore di onda quadra a livello TTL (0 ÷ 3V). Verificare ingresso ed uscita su di uno oscilloscopio, al variare dei parametri di DELAY.

- Realizzare generatori di impulsi con programmabilità di ritardo, durata e numero impulsi. Verificare il funzionamento del programma su oscilloscopio.

Tabella IX - Istruzioni di BIT, RES, SET.

BIT	40	CB,. + reg + bit
RES	80	
SET	C0	

reg	B	C	D	E	H	L	(HL)	A
bit	0	1	2	3	4	5	6	7
	00	08	10	18	20	28	30	38

Per ottenere il codice operativo completo occorre sommare al codice della funzione i codici di registro e di bit.

Esempio:

RES 6, H : CB, xx

con xx = $80_H + 4_H + 30_H = B4_H$

codice RES
registro H
bit 6

9. Esercizi di riepilogo

Esercizio 9.1 - Contatore su due cifre

Questo programma fa comparire sul visualizzatore due cifre ed incrementa la meno significativa con riporto sulla più significativa. Le cifre sono esadecimali, quindi comparirà la sequenza 00,01...0E,0F,10.... 1F,20... FF,00... Il programma è formato di due parti:

- il contatore vero e proprio (CONT)
- il sottoprogramma di visualizzazione (DISP)

Quest'ultimo porta sul display, in forma di due cifre esadecimali, il contenuto dell'accumulatore. Può essere utilizzato anche da altri programmi.

Programmi 9.1

800	21,00,0C	CONT	LD HL, 0C00	puntatore al display
803	06,00		LD B, 00	B è usato come contatore
805	78	LOOP	LD A,B	prepara in A la parola da visualizzare in esadecimale
806	CD,10,08		CALL DISP	salto al programma di visualizzazione
809	CD,87,01		CALL RATE P8	ritardo
80C	04		INC B	incrementa il contatore
80D	C3,05,08		JMP LOOP	

Il sottoprogramma DISP viene chiamato con:
- indirizzo del display (0C00) in HL.
- dato da visualizzare in A.

810	F5	DISP	PUSH A	Salva A nello stack
811	E6,0F		AND 0F	Maschera i 4 bit bassi
813	77	OUT1	LD (HL),A	Porta sul display (cifra in posizione 0) il contenuto di A
814	F1		POP A	Ripristina A
815	1F		RRA	} Scalano i 4 bit alti di A nei 4 bit bassi.
816	1F		RRA	
817	1F		RRA	
818	1F		RRA	
819	E6,0F		AND 0F	Maschera
81B	C6,10		ADD 10	punta alla seconda cifra da destra
81D	77	OUT2	LD (HL),A	Porta sul display A
81E	C9		RET	

Una volta caricato e lanciato il programma si può verificare che opera correttamente incrementando il contatore, ma che una cifra appare piú illuminata dell'altra. Questo perchè il ritardo è stato inserito una sola volta nell'anello del programma e quindi le due operazioni di OUT non sono equispaziate nel tempo (vedi Fig. 2.24).

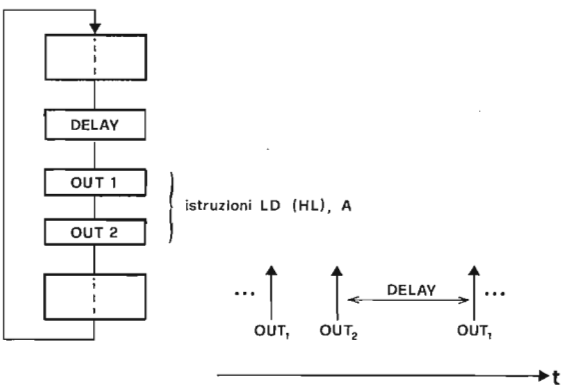


Figura 2.24

La cifra presentata con OUT2 resta sul display molto piú a lungo. Per ottenere la stessa intensità sulle due cifre occorre inserire un DELAY anche tra OUT1 e OUT2.

810	F5	DISP	PUSH A
811	E6,0F		AND 0F
813	77		LD (HL),A
814	CD, ,		CALL DELAY inserire ritardo
817	F1		POP A

Esercizio consigliato

Realizzare un contatore su 4 cifre, partendo da una coppia di registri e scrivendo due sottoprogrammi DISP, rispettivamente per 1°, 2° e 3°, 4° cifra. (Per il secondo DISP basta predisporre nel registro A 2X e 3X per il trasferimento su CONADD).

Uso del sottoprogramma di visualizzazione del monitor

Il programma di Monitor Picomon-V1 è costituito da un programma principale e dalle routine di esecuzione dei vari comandi[1]. Un diagramma di flusso semplificato è in Fig. 2.25.

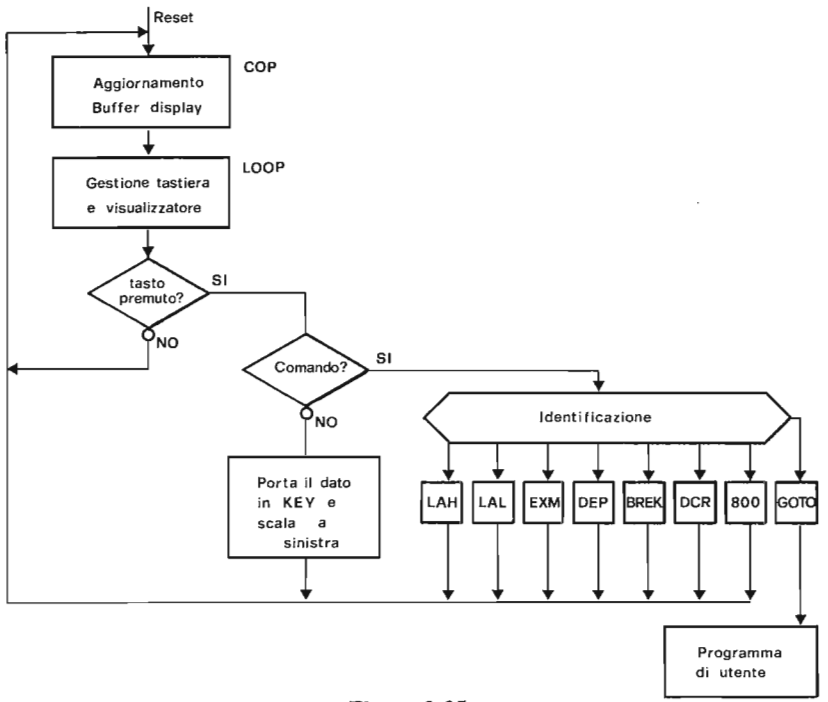


Figura 2.25

Il modulo LOOP provvede a visualizzare sul display il contenuto del buffer BF0/1/2/3 (Vedi pag. 10), gestisce la tastiera introducendo i nuovi dati nel campo KEY e determina i salti ai sottoprogrammi di esecuzione dei comandi.

LOOP è organizzato in modo tale da poter essere utilizzato come sottoprogramma di utente. Le specifiche sono: sottoprogramma LOOP indirizzo di chiamata: 0044_H.

(1) Per maggiori dettagli vedi appendice G.

Usa ed altera tutti i registri. Determina la visualizzazione sul display delle celle 0BF0/1/2/3. Durante la chiamata di LOOP da parte di un programma di utente sono mantenute le funzioni di monitor (comandi e campo dati).
Vediamo ad esempio la sequenza di operazioni eseguita dal programma:

```
USER   CALL   LOOP
       JP     USER
```

Premendo un tasto qualsiasi si esce da LOOP, viene eseguita l'operazione corrispondente e si rientra in LOOP. Dato che in questo modo non viene mai chiamato il programma COP che aggiorna il buffer display; i comandi sono eseguiti, ma non se ne vedono i risultati.

Per verificare questo fatto scriviamo il programma precedente a 800.

800 803	CD,44,00 C3,00,08	USER	CALL LOOP JP 800

Dopo aver dato GOTO a 800 il contenuto del display non è alterato. Possiamo eseguire tutte le operazioni di tastiera, ad esempio la sequenza:

```
00     LAL
09     LAH
11     DEP
22     DEP
33     DEP
```

che carica da 900 in avanti 11, 22, 33. Sul display comparirà sempre 800; uscendo dal programma e rientrando sul monitor con un RESET possiamo però verificare che le operazioni di scrittura sono state effettivamente eseguite.

Gli esercizi seguenti sono esempi di uso di LOOP in programmi di utente.

Esercizio 9.2

In questo esercizio si realizza un contatore esadecimale sul display sfruttando la routine LOOP. Da notare che sul display compariranno sempre otto cifre, anche se il contatore ne utilizza di meno.

Programma 9.2: Contatore su cifre 0 e 1.

800	21,F0,0	CONT	LD HL,	puntatore al buffer display
803	34		INC (HL)	incrementa il puntatore
804	CD,44,00		CALL LOOP	visualizzazione
807	CD,87,01		CALL RATEP8	ritardo
80A	C3,00,08		JP CONT	chiude l'anello

Il conteggio inizia dal valore contenuto precedentemente in BF0; le altre cifre (dalla 3° alla 7°) restano immutate.

Dato che il ritardo è inserito dopo LOOP, l'ultima cifra visualizzata (cioè quella piú a sinistra), rimane accesa per piú tempo ed appare piú luminosa delle altre.

Contatore su N cifre

Il programma 9.2 è facilmente estendibile a tutte le cifre del display. Occorre incrementare BF1 quando BF0 passa da FF a 00 (cioè genera un riporto) e cosí via. Costruiamoci una subroutine che esegue questa funzione; verrà chiamata ogni volta che si genera un riporto dal conteggio.

810	23	INCR	INC HL	passa alla cifra adiacente a sinistra sul display (da BUFF a BUFF + 1)
811	34		INC (HL)	la incrementa
812	CC,10,08		CALL Z INCR	se il risultato è 00 c'è un nuovo riporto sulla cifra successiva
815	2B		DEC HL	riporta il puntatore al valore originale
816	C9		RET	

Questa stessa subroutine INCR può essere usata per l'operazione di incremento entro il programma principale CONT (usando un CALL incondizionato). Unica avvertenza è predisporre HL su BUFF -1, perchè viene subito incrementato da INC HL.

Programma 9.3. Contatore su 8 cifre.

800	21,EF,0B	CONT	LD HL, EF	carica il puntatore a BUFF -1
803	CD,10,08		CALL INCR	incrementatore
806	CD,44,00		CALL LOOP	visualizzazione
809	C3,00,08		JP 800	chiude l'anello

Questa volta, non essendovi ritardi in particolari posizioni, le cifre appaiono tutte ugualmente illuminate. Notiamo una particolarità: la subroutine INCR può chiamare se stessa anche piú volte.

Esercizio consigliato

- Scrivere un sottoprogramma di inizializzazione che azzeri il buffer BF0/1/2/3 e collocarlo in testa a CONT per far partire il contatore da 00000000.

Contatore decimale - Istruzione DAA

Per realizzare un contatore decimale di N cifre, visualizzato sul display, possiamo utilizzare la stessa struttura del programma 9.2, limitando però a 9 il valore di ciascuna cifra.

Questa operazione viene automaticamente eseguita dall'istruzione DAA (Decimal Adjust Accumulator). Quando un semibyte dell'accumulatore contiene un numero maggiore di 9 (ad esempio A, B.... F), gli viene sommato 6. Questo genera un riporto intermedio ed "aggiusta" il contenuto dell'accumulatore al codice BCD.

Esercizio 9.3:

Contenuto di A = prima	Dopo DAA
15	15
1A	26
	$(A_H + 6) = 16$

L'istruzione DAD opera correttamente se è preceduta da un ADD o SUB. Modifichiamo quindi, in base a quanto sopra indicato, la routine INCR:

810	23	INCR	INC HL
811	7E		LD A, (HL)
812	C6, 01		ADD A 01
814	27		DAA
815	77		LD (HL), A
816	CC,10,08		CALL Z INCR
819	2B		DEC HL
81A	C9		RET

Questo sottoprogramma, abbinato all'esercizio 9.2, realizza sul visualizzatore un contatore decimale su otto cifre.

Esercizio 9.4 Misuratore di tempo

Utilizzando il contatore realizzato negli esercizi precedenti possiamo ora ottenere un misuratore di tempi / cronometro. Per questo è sufficiente aggiungere un programma di inizializzazione, che azzeri il contenuto del buffer display all'arrivo del programma, e modificare l'incrementatore inserendo un test che condiziona l'incremento allo stato di un bit di ingresso. La cifra indicata è in questo modo proporzionale al tempo per cui il bit rimane, ad esempio, a livello "1" logico.

Inizializzazione del buffer:

830	21,F0,0B	INIZ	LD HL, BUFF	puntatore al buffer
833	36,00		LD (HL), 0	prima cella a 0
835	23		INC HC	
836	36,00		LD (HL), 0	seconda cella a 0
838	23		INC HL	
839	36,00		LD (HL), 0	
83B	23		INC HL	
83C	36,00		LD (HL), 0	ultima cella a 0
83E	C9		RET	

Incrementatore modificato:

810	3A.000C	INCT	LD A,(CO-NADD)	legge registro ingresso
813	CB, 5E		BIT INPUT, A	testa bit 3
815	CA,22,08		JPZ ENDINC	se zero non incrementa
818	23	INCR	INC HL	}
819	7E		LD A, (HL)	}
⋮	⋮		⋮	} come esercizio 9.3
821	2B		DEC HL	}
822	C9	ENDINC	RET	}

Il programma principale è lo stesso dell'esercizio 9.2, ma alla partenza chiama INIZ.

800	CD,30,08	TMIS	CALL INIZ	inizializzazione
803	21,EF,0B	MISLOP	LD HL, BEF_H	puntatore a buffer -1
806	CD,10,08		CALL INCT	incrementa e test
809	CD,44,00		CALL LOOP	visualizza
	C3,03,08		JP MISLOP	chiude l'anello

Il bit INPUT è un segnale elettrico TTL proveniente dall'esterno tramite il connettore B. Invertendo la condizione di salto a ENDINC si misura il periodo per cui un segnale rimane a 0.

Questo esercizio è suscettibile di molte varianti, interessanti dal punto di vista didattico ed applicativo.

- Misura diretta del tempo in secondi e decimi: inserire entro INC aver routine di ritardo opportuno (vedi paragrafo 5), tale che l'intervallo tra un incremento e l'altro della cifra meno significativa sia pari a 1/10 di secondo.
- Misura dell'intervallo di tempo di tempo fra due impulsi su linee di ingresso distinte.

Esercizio 9.5 - Convertitore D/A a parzializzazione

Obiettivo è controllare da programma una corrente od una tensione analogica, allo scopo di variare la potenza fornita ad un carico (per esempio un piccolo motore o una lampadina). La tecnica usata è quella della parzializzazione, indicata in Figura 2.26

Il problema richiede il progetto di una interfaccia per l'organo "di potenza" e la stesura del programma di gestione. Quest'ultimo a sua volta comprenderà il modulo di controllo vero e proprio ed un modulo di colloquio con l'operatore, che permette di inserire i parametri di controllo. Per quest'ultimo useremo sottoprogrammi già esistenti nel "monitor".

Come interfaccia con l'organo attuatore esterno, se ci limitiamo a basse tensioni ed a correnti dell'ordine di qualche centinaio di mA, sono sufficienti normali porte di potenza (buffer) TTL con uscita a collettore aperto. Per non interferire con la gestione della tastiera e del display, useremo le due uscite impulsive PUL 1 e PUL 2, rispettivamente come set e reset di un flip-flop che genera direttamente la forma d'onda di comando della parzializzazione. Lo schema dell'interfaccia esterna è in Figura 2.27.

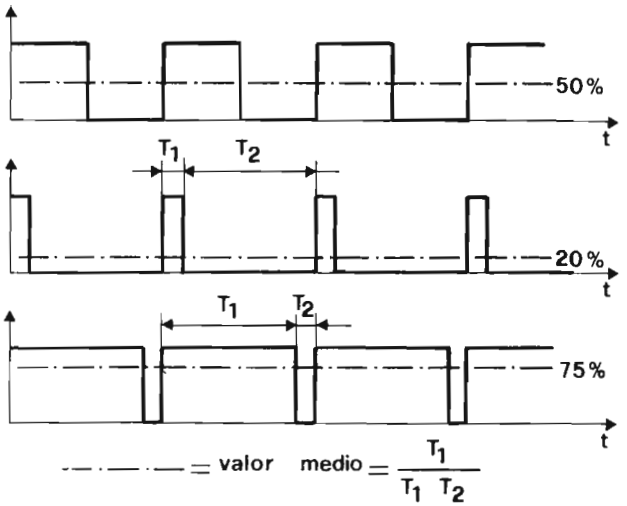


Figura 2.26 - Tecnica della parzializzazione.

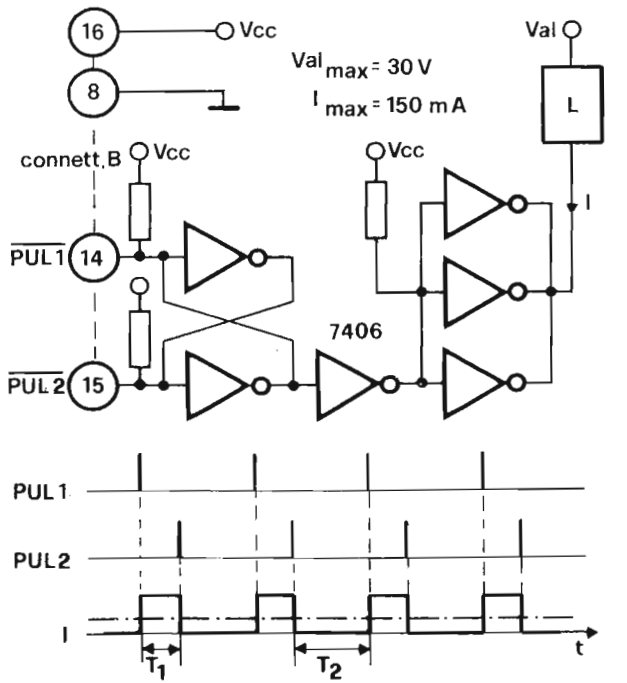


Figura 2.27 - Interfaccia per controllo di piccoli carichi.

Il carico L (motore o lampadina) ha costanti di tempo molto maggiori della durata degli impulsi di corrente T_1 e del loro intervallo di ripetizione T_2 pertanto agisce come filtro passo - basso e media la potenza fornita dagli impulsi.

Variando il rapporto T_1 / T_2 varia il duty-cycle dell'onda quadra e quindi il valore medio della corrente I nel carico L. La struttura del programma di controllo è molto semplice (v. Figura 2.28).

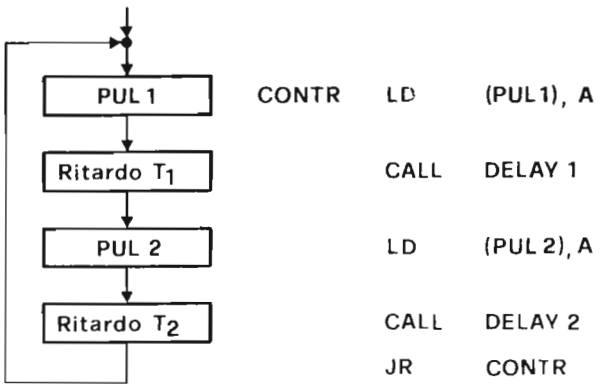


Figura 2.28 - Strutture del programma di controllo.

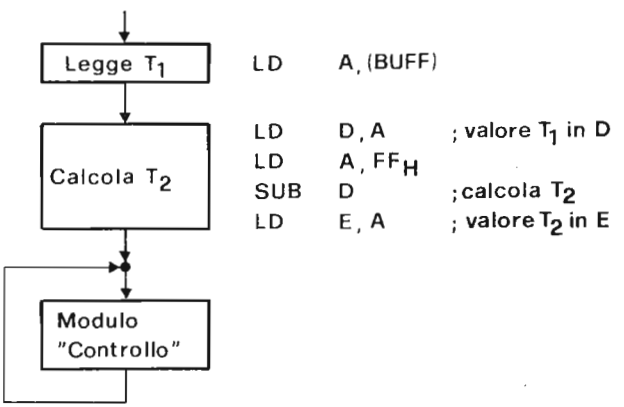


Figura 2.29 - Controllo con parametro fisso. Il modulo "CONTROLLO" usa il contenuto dei registri D ed E come parametro di DELAY.

Le ultime due cifre introdotte dal campo dati della tastiera vengono poste nella cella BUFF; è quindi comodo stabilire che il valore presente in questa cella rappresenta il valore di T_1.

Per quanto riguarda T_2 possiamo scegliere un valore costante, oppure mantenere costante il periodo totale $T_1 + T_2$ calcolando T_2 a partire da T_1. Questa seconda soluzione permette di variare la potenza nel carico da 0 al 100%. Il massimo valore di T_1 è FF_H, quindi:

$$T_2 = FF_H - T_1$$

Il parametro di parzializzazione (cioè il valore di T_1) può essere dato una volta per tutte prima del lancio del programma; in questo caso la sequenza di operazioni è quella di Figura 2.29. Diversamente possiamo scegliere una soluzione maggiormente interattiva, che permetta di modificare il parametro T_1 anche mentre il programma è in esecuzione. Possiamo per questo usare il blocco SCAN del monitor, che trasferisce le cifre via via introdotte dalla tastiera nella cella BUFF. (Il diagramma di flusso in questo caso è riportato in Figura 2.30).

Il listato completo del programma, assemblato per il Picocomputer, è LIST 1.

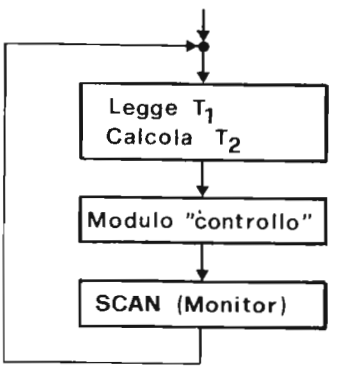


Figura 2.30 - Controllo con parametro variabile. Il blocco SCAN (presente nel Monitor) aggiorna BUFF e gestisce tastiera e visualizzatore.

Esercizio 9.6 Misura di periodo e conversione A/D

Questo esempio è il "duale" del precedente; obiettivo è infatti convertire un tempo in un valore numerico. Possiamo così misurare direttamente il periodo di un segnale, oppure, per via indiretta, tensioni, correnti, resistenze. In questi casi l'interfaccia esterna sarà un convertitore tensione (o corrente) / frequenza, realizzato ad esempio secondo le tecniche descritte in *La progettazione dei circuiti Amplificatori Operazionali, pag. 9-22* (Ed. Gruppo Editoriale Jackson).

Per questo particolare esempio faremo riferimento ad un semplicissimo convertitore resistenza/periodo (v.Figura 2.31). Questo circuito permette di realizzare con gli opportuni trasduttori, misuratori di luce, temperatura, e così via. Grazie alla presenza del microprocessore, eventuali linearizzazioni possono essere eseguite da programma,

```
                                1  ;
                                2  ;
0800                            3          ORG     0800H
                                4  ;
                                5  ;
                                6  ;       CONVERTITORE D/A A PAR. TAL.IZZAZIONE
                                7  ;
                                8  ;
                                9  ;
                               10  ;       DEFINIZIONE PARAMETRI
                               11  ;
                               12  BUFF    EQU     0BFOH
                               13  PUL1    EQU     100H
                               14  PUL2    EQU     500H
                               15  COP     EQU     44H
                               16  USCI    EQU     0FD0H
                               17  ;
                               18  ;
                               19  ;
0800   31D00F                  20  CONV    LD      SP,USCI
0803   2AF00B                  21  CONV1   LD      A,(BUFF)
0806   57                      22          LD      D,A
0807   3EFF                    23          LD      A,0FFH
0809   ED                      24          SUB     D
080A   5F                      25          LD      E,A
080B   210001                  26          LD      (PUL1),A
080E   47                      27          LD      B,D
080F   10FE                    28  LOP1    DJNZ    LOP1
0811   ED5002                  29          LD      (PUL2),E
0814   43                      30          LD      B,E
0815   10FE                    31  LOP2    DJNZ    LOP2
0817   CD4400                  32          CALL    COP
081A   18E7                    33          JR      CONV1
                               34  ;
                               35  ;
                               36  ;
                               37  ;
```

LIST 1

usando adatti algoritmi o tabelle. Dato che la precisione ottenibile con queste tecniche di conversione è limitata dalla parte analogica, limitano la misura ad una risoluzione di 8 bit.

Il "programma di acquisizione" deve pertanto misurare l'intervallo di tempo T (v.Figura 2.31), compreso tra due transizioni omologhe (0-1 oppure 1-0) del segnale U riportato su una linea di ingresso. Dal momento che sono sufficienti 8 bit, useremo un registro come contatore, incrementato periodicamente durante l'intervallo T. La costante di tempo RC va scelta in modo da non superare 255 cicli (FF_H) per il massimo valore di R.

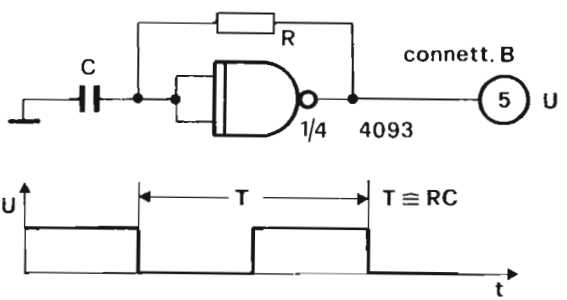


Figura 2.31

Valori adatti sono R_{max} = 500 KΩ, C = 10 nF. Le operazioni da compiere sono indicate nella Figura 2.32.

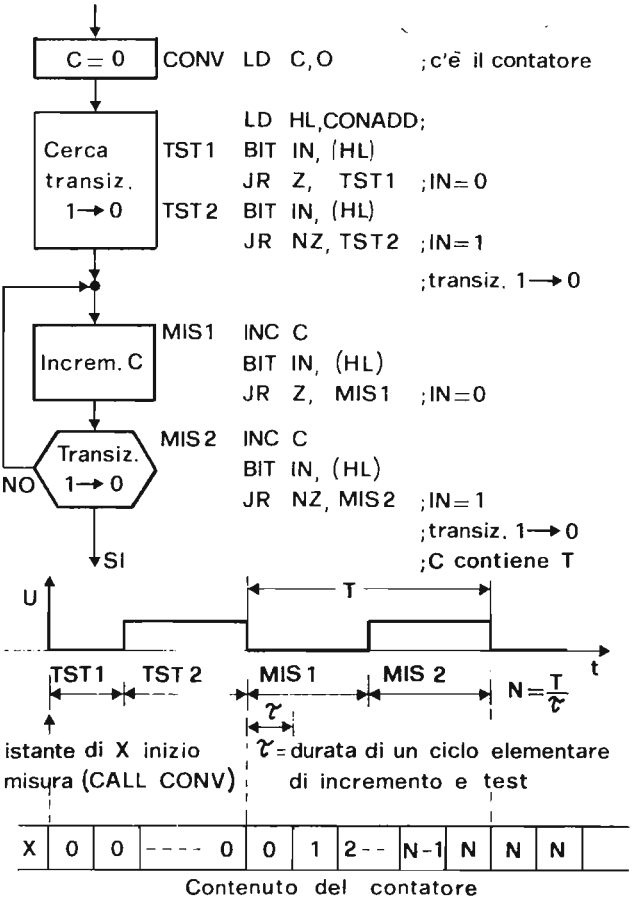


Figura 2.32 - Misura del periodo di un segnale U mediante conteggio di cicli elementari. "N" è il risultato della conversione A/D.

Dobbiamo anche visualizzare il contenuto del registro contatore; trattandosi di due sole cifre esadecimali, non conviene usare il sottoprogramma SCAN del monitor, che scandisce tutte le 8 cifre del display. Prepareremo pertanto una routine apposita, che diventa il programma principale per questa applicazione e chiama il modulo "Misura periodo" per aggiornare il registro contatore.

Nel programma (v. figura 2.33) il modulo di conversione viene chiamato due volte, ma si visualizza solo il risultato della prima conversione. La seconda chiamata ha il solo scopo di mantenere accese le due cifre per uno stesso tempo, perchè abbiano la stessa luminosità.

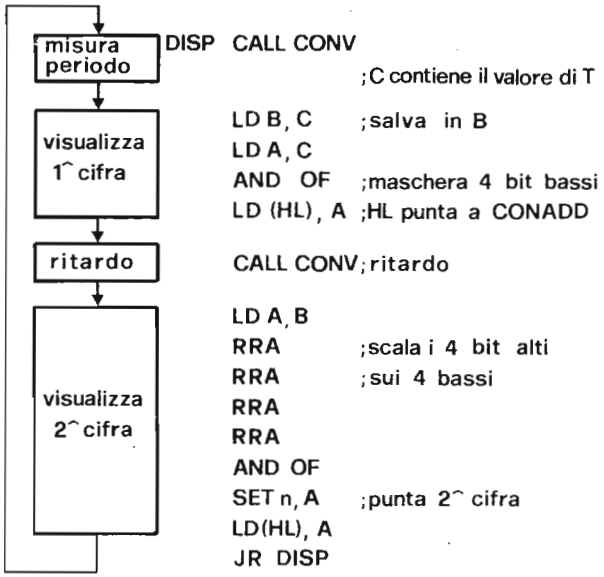


Figura 2.33 - Misura di periodo e visualizzazione

Il formato della parola da inviare alla tastiera per la gestione del display è in Figura 2.34.

Il listato completo per questa applicazione è LIST 2.

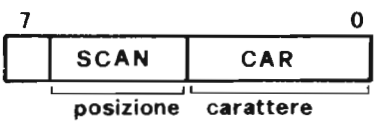


Figura 2.34 - Formato per la presentazione di un carattere sul display.

Esercizio 9.7 - Misuratore di tempo

Obiettivo è realizzare un contatore su 8 cifre e farne comparire il contenuto sul visualizzatore della tastiera. Il contatore verrà incrementato ad intervelli di tempo prefissati, sotto controllo di alcuni flag esterni, ad esempio con funzioni di:

- abilitazione al conteggio (ENable);
  azzeramento (ReSet);
  elezione incremento/decremento (Up/Down).

Il contatore può essere configurato per presentare ore, miniti primi e secondi, decimi e centesimi.

```
                        1  ;
                        2  ;
                        3          CONVERTITORE A/D
                        4  ;
                        5  ;
                        6          PARAMETRI
                        7
                        8  USP     EQU     0ED0H
                        9  CONADD  EQU     0C00H
                       10  IN      EQU     4       ;PIN 5 CONN. E
                       11  ;
                       12  ;
                       13  ;
0800                   14          ORG     800H
                       15  ;
                       16  ;
0800   211D00F         17  DISP    LD      HL,     USP
0803   CD1A08          18          CALL    CONV
0806   41              19          LD      B,      C
0807   79              20          LD      A,      C
0808   E60F            21          AND     0FH
080A   77              22          LD      (HL),A
080B   CD1A08          23          CALL    CONV
080E   79              24          LD      A,      C
080F   1F              25          RRA
0810   1F              26          RRA
0811   1F              27          RRA
0812   1F              28          RRA
0813   E60F            29          AND     0FH
0815   CB70            30          SET     6,      A
0817   77              31          LD      (HL),A
0818   18F8            32          JR      DISP1
                       33  ;
                       34  ;
                       35  ;
                       36          MISURA DEL PERIODO
                       37  ;
081A   0E00            38  CONV    LD      C,      0
081C   21000C          39          LD      HL,     CONADD
081F   CB66            40  TST1    BIT     IN,     (HL)
0821   28FC            41          JR      Z,      TST1
0823   CB66            42  TST2    BIT     IN,     (HL)
0825   20FC            43          JR      NZ,     TST2
0827   0C              44  MIS1    INC     C
0828   CB66            45          BIT     IN,     (HL)
082A   28FB            46          JR      Z       MIS1
082C   CB56            47  MIS2    BIT     IN,     (HL)
082E   20FB            48          JR      NZ,     MIS2
0830   C9              49          RET
                       50  ;
                       51  ;
                       52  ;
```

LIST 2

Le operazioni da eseguire sono raggruppabili in due moduli di base: il contatore vero e proprio e la routine di visualizzazione (v. Figura 2.35).

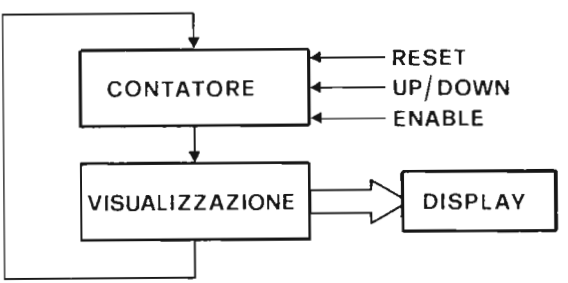


Figura 2.35 - Funzioni del misuratore di tempo.

Per quest'ultima utilizzeremo una parte del programma di monitor, e precisamente il modulo COP, che gestisce la visualizzazione sul display del contenuto di quattro celle di memoria, da BUFF a BUFF + 3. (vedi BIT n° 3 pag. 40 e BIT n° 5 pag. 48). Il modulo "contatore" deve pertanto intervenire sul contenuto delle celle da BUFF a BUFF + 3. La cella BUFF, che corrisponde alla cifra piú a destra, sarà incrementata ad ogni ciclo; le successive solo in caso di riporto.

Un esempio di programma che esegue queste operazioni è in Figura 2.36.

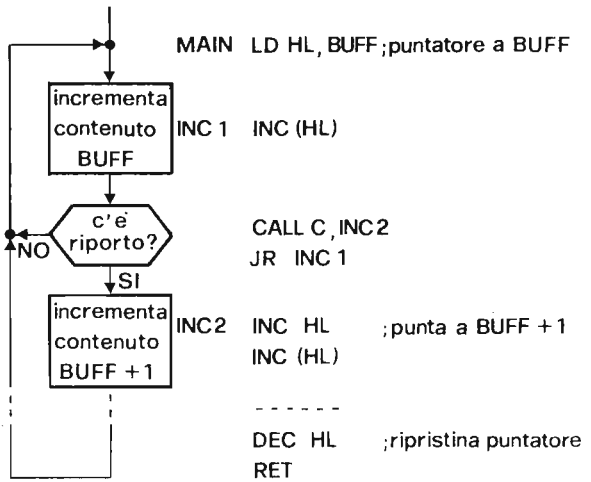


Figura 2.36

Al posto della riga puntinata possiamo inserire una chiamata condizionata alla routine che incrementa BUFF + 2 in caso di riporto su BUFF + 1, e cosí via. Osservando però che tutti questi moduli eseguono essenzialmente la stessa sequenza di operazioni (predispongono la coppia di registri HL come puntatore alla memoria ed incrementano il contenuto della cella puntata), possiamo compattarli in un unico sottoprogramma che, in caso di riporto, richiama se stesso.

```
MODULO "INCREMENTATORE"
INCA    INC HL          ;puntatore a BUFF successivo
        INC (HL)        ;ne incrementa il contenuto
        CALL C INCA     ;se c'è riporto ripete l'operazione
        DEC HL          ;ripristina il puntatore originale
        RET
```

Queste stesse operazioni vengono scorporate anche dal programma principale, che, inserendo anche la chiamata della routine di visualizzazione, diventa:

```
MAIN    LD HL,BUFF-1    ;predispone il puntatore
        CALL INCA       ;incrementatore
        CALL COP        ;visualizzazione
        JR MAIN
```

Questo modulo, insieme all'incrementatore, implementa già un contatore esadecimale su 8 cifre, visualizzato sul display della tastiera.

Per soddisfare le specifiche iniziali occorre ancora testare i segnali di ingresso per decidere se eseguire o meno un azzeramento, se abilitare il conteggio, se incrementare o decrementare. Per questo è sufficiente inserire nel modulo "incrementatore" una serie di operazioni condizionate dal test sul corrispondente comando (dato attraverso linee di ingresso). Il nuovo diagramma di flusso è in Figura 2.37.

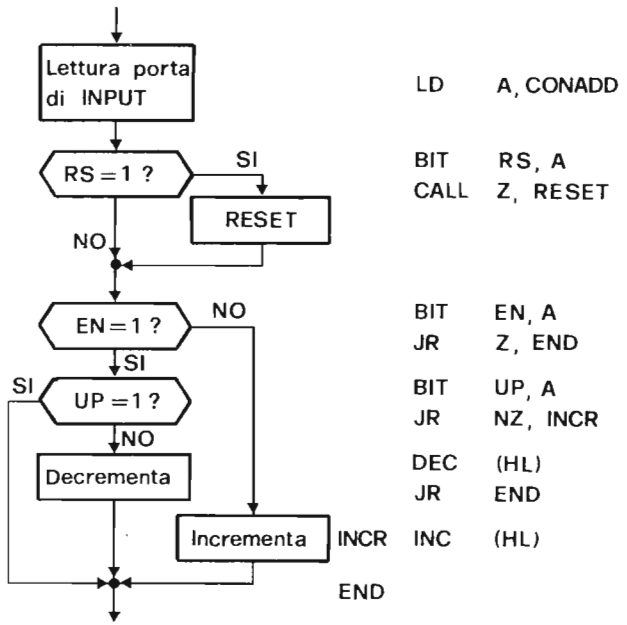


Figura 2.37 - Comandi al contatore

Il contatore a questo punto conta, e può essere bloccato o resettato.

Ciascuna cella di BUFF assume i valori da 0 a 255, visualizzati sulla tastiera con due cifre esadecimali. Per ottenere un contatore decimale o secondo altri moduli (per esempio modulo 60 per un orologio con minuti primi e secondi), occorre inserire altri blocchi, rispettivamente di conversione binario - decimale e di limite all'incremento di una cella. Per la conversione binario - decimale è comodo usare l'istruzione Decimal Adjust Accumulator (DAA; per i dettagli sulle operazioni eseguite vedi manuale Z80 o 8080), che però opera solo sull'accumulatore e deve essere preceduta da una istruzione ADD per operare correttamente. All'istruzione INC (HL) sostituiremo quindi:

```
LD A, (HL)    ;trasferisce nell'accumulatore
ADD 01        ;incrementa
DAA           ;conversione binario-decimale
LD (HL), A    ;ritrasferisce in BUFF
```

In coda a questo blocco possono essere inserite le istruzioni per il conteggio su modulo MOD diverso da 100:

```
CP MOD                  ;conteggio massimo?
CALL Z OVERMOD          ;si, dà riporto
...                     ;no, continua
OVERMOD LD (HL), 00     ;azzera
CALL INC                ;riporto su cifra
RET                     ;successiva
```

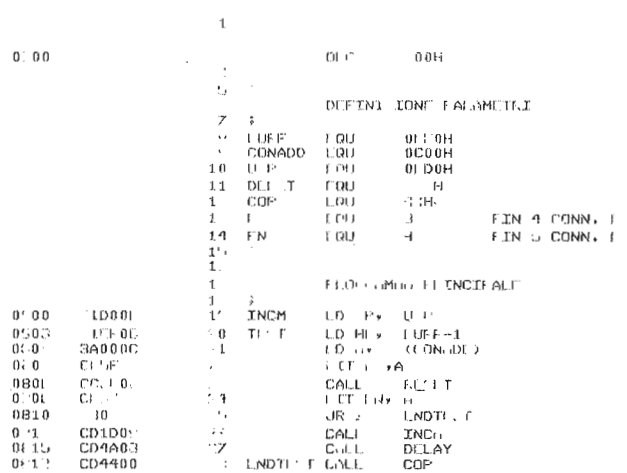
Perchè il misuratore di tempo diventi completamente operativo occorre ancora tararlo, cioè definire l'intervallo di tempo tra un incremento e l'altro. Questo tempo può essere calcolato dal programma completo, determinando il numero di cicli impiegati per eseguire un loop del programma principale. Volendo invece un contatore direttamente tarato, ad esempio in centesimi di secondo (e questa è la nostra specifica), è sufficiente inserire una routine di ritardo con parametro opportuno.

Va messo in evidenza che la precisione ottenibile è limitata, perchè a causa delle istruzioni di salto condizionato il numero di cicli richiesti dal programma non è un dato costante. Ad esempio, in caso di riporto su una cella del contatore si deve incrementare la successiva, e questo comporta l'esecuzione di un maggior numero di istruzioni.

È possibile eliminare questo errore introducendo in posizione opportune delle istruzioni che hanno il solo scopo di eguagliare i tempi di esecuzione per qualsiasi situazione. L'errore introdotto diventa comunque trascurabile se l'intervallo di incremento è determinato in massima parte dalla routine di ritardo, che viene sempre eseguita allo stesso modo.

Il listato completo del programma che implementa un contatore tipo "cronometro", con reset ed abilitazione è riportato come LIST 3.

Si può notare come siano presenti due moduli incrementatori distinti, rispettivamente per frazioni di secondo (modulo 100) e per ore, primi e secondi (modulo 60). Durante l'esecuzione del programma si noterà anche che le cifre non sono uniformemente illuminate; questo perchè durante il ritardo (cioè per la maggior parte del tempo), viene visualizzata sempre la stessa cifra (ultima a sinistra), che diventa quindi piú luminosa.



```
0811    1E..            29              JR      TF..I
                        30      ;
                        31
                        32      ;
                        33              INCREMENTATORE PER CENTESIMI E DECIMI
                        34
081D    23              35      INCA    INC     HL
081E    7E              36              LD A,   (HL)
081F    C601            37              ADD A, 01
0821    27              38              DAA
0822    77              39              LD (HL),A
0823    DC2908          40              CALL C  INC I
0826    2B              41              DEC HL
0827    C9              42              RET
                        43
                        44
                        45      ;
                        46      ;       INCREMENTATORE PER SECONDI, PRIMI E ORE
                        47      ;
0829    23              48      INC I   INC     HL
082A    7E              49              LD A,   (HL)
082B    C601            50              ADD A,  01
082D    27              51              DAA
082E    77              52              LD (HL),A
082F    FE60            53              CP      60H
0831    CC3408          54              CALL Z  INC_C
0834    2B              55              DEC HL
0835    C9              56              RET
                        57
                        58      ;
                        59
                        60
0836    3600            61      INC_C   LD (HL),0
0838    CD2908          62              CALL    INC I
083B    C9              63              RET
                        64
                        65
                        66      ;       AZZERAMENTO DEL CONTATORE
                        67      ;
083C    E5              68      RESET   PUSH HL
083D    21F000          69              LD HL,  BUFF
0840    3E00            70              LD A,   00
0842    77              71              LD (HL),A
0843    23              72              INC     HL
0844    77              73              LD (HL),A
0845    23              74              INC     HL
0846    77              75              LD (HL),A
0847    23              76              INC     HL
0848    77              77              LD (HL),A
0849    E1              78              POP     HL
084A    C9              79              RET
                        80
                        81
                        82      ;
                        83      ;       RITARDO PER TEMPORIZZAZIONE
                        84      ;
084B    C5              85      DELAY   PUSH    BC
084C    0164 07         86              LD BC,  DELAY
084F    0B              87      DELOP   DEC     BC
0850    78              88              LD A,   B
0851    B1              89              OR      C
0852    20FB            90              JR NZ   DELOP
0854    C1              91              POP     BC
0855    C9              92              RET
                        93      ;
                        94      ;
```

LIST 3

10. Gestione delle interruzioni

L'interruzione è una chiamata a sottoprogramma determinata, anzichè dalla esecuzione di un CALL, dalla attivazione di un apposito segnale di controllo della CPU (Interrupt Request).

Lo Z80 possiede diversi "modi" di gestione delle interruzioni, nel complesso piú sofisticati rispetto ad altri microprocessori ad 8 bit; per sfruttare appieno queste possibilità il periferico interrompente deve poter inviare alla CPU delle informazioni che permettono di scegliere il sottoprogramma da eseguire (vettore di interrupt).

Nel caso del PICOCOMPUTER è possibile usare solo il piú semplice dei modi di interruzione (Modo 1) che non prevede vettore ed equivale ad un CALL alla cella 38_H (RST 38_H).

Questa cella è in ROM e contiene una istruzione di salto alla cella 0BF5 in RAM. In quest'ultima può essere scritto il salto all'effettivo programma di servizio dell'interruzione preparato dall'utente.

L'interruzione è accettata solo se la CPU ha ricevuto un comando "Enable Interrupt" (istruzione EI). Quando la CPU ha accettato l'interruzione emette un segnale di riconoscimento. (Interrupt Acknowledge: INA) che deve essere usato per rimuovere la richiesta di interruzione.

Un semplice circuito da collegare al PICO e che permette di dare una interruzione (ed una sola) su comando esterno, è in fig. 2.38.

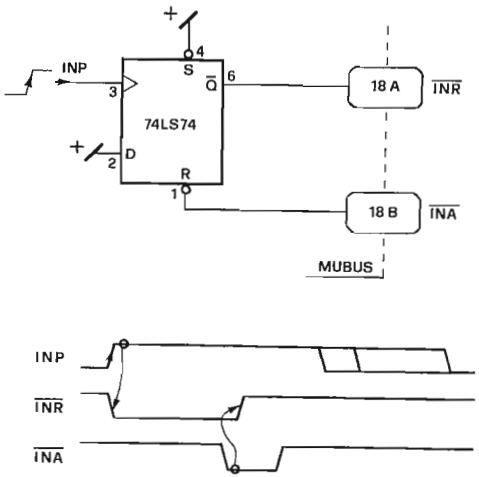


Figura 2.38

Esercizio 10

Questo programma permette, unitamente al circuito di cui sopra, di dare dall'esterno il comando EXM del monitor, tramite interruzione.

800 802 803	ED, 56 FB C3,00,00	E 10	IM 1 EI JP,0	predispone il modo 1 abilita le interruzioni ritorna al monitor
0BF5	C3,E9,00	INTEXEC	JP EXM	salto al comando EXM del monitor

Attivando INP si esegue un EXM.
Se il comando è dato a mano occorre inserire un circuito antirimbalzo.
L'esecuzione dell'interruzione disabilita immediatamente l'accettazione di ulteriori Interrupt.

Per dare un nuovo comando EXM sotto interrupt occorre quindi dare prima un GO-TO a questo programma.

APPENDICE A

TABELLA COMPLETA DELLE ISTRUZIONI Z80 E MODO DI IMPIEGO

La tabella raccoglie e completa quelle già presenti nel testo, al termine di ogni gruppo di esercizi.

Essa comprende tutti i codici operativi eseguiti dal microprocessore Z80, raccolti su un'unica pagina. (1) Il compattamento sfrutta alcune caratteristiche dei codici; in particolare per le istruzioni che usano i registri IX e IY viene indicato una volta per tutte il prefisso da applicare alla corrispondente istruzione con i registri HL. Gruppi di istruzioni con lo stesso prefisso sono riuniti in uniche tabelle; il prefisso da usare è specificato a lato della tabella ed indicato con un asterisco (*) nel codice operativo. Le istruzioni di shift, rotate, bit test, set e reset hanno una struttura completamente modulare e sono raggruppate in una tabella molto compatta che però richiede l'esecuzione di una o due addizioni per ottenere il codice operativo completo. Anche le istruzioni di LoaD potrebbero essere riunite in poche caselle con la stessa tecnica, ma si è preferito tenerle distinte perchè di uso molto frequente.

La tabella non sostituisce il manuale con la spiegazione completa delle istruzioni, anzi, una certa familiarità con questo è necessaria per usarla con scioltezza. È comunque un comodo ausilio di consultazione rapida che evita di dover sfogliare pagine e pagine durante il debug di un programma o per assemblare a mano poche istruzioni.
I colori indicano diverse classi di istruzioni:

Verde indica operazioni eseguite anche con i registri IX ed IY anteponendo i prefissi DD e FD rispettivamente. Queste istruzioni con i registri indice comprendono solo operazioni dirette sui registri.

Giallo indica operazioni eseguite con i registri IX ed IY in cui però il registro è usato come puntatore alla memoria. In questo caso occorre anche specificare lo scostamento d. Il formato completo è:

<prefisso>, <cod. op. per HL (I byte)>, <scostamento>, [cod. op. (II byte)]
in cui il campo [ ] è opzionale.

Arancione indica altre istruzioni che accettano il prefisso e lavorano sui registri IX ed IY ma non sono riportate dai manuali (vedi articoli su Bit n°3).

I valori forniti come dato immediato sono indicati dai simboli:

n : parole di 8 bit (istr. di trasferimento e aritmetiche),
nn: parole di 16 bit (istr. di trasferimento e di salto),
d : scostamento (8 bit) per i registri indice,
e : scostamento (8 bit) per istruzioni di salto relativo.

(1) La tabella completa delle istruzioni Z80, è riportata in IV^a di copertina.

Esempi di uso

Per indicare la tabella via via utilizzata si fa riferimento alla fig. A.1.

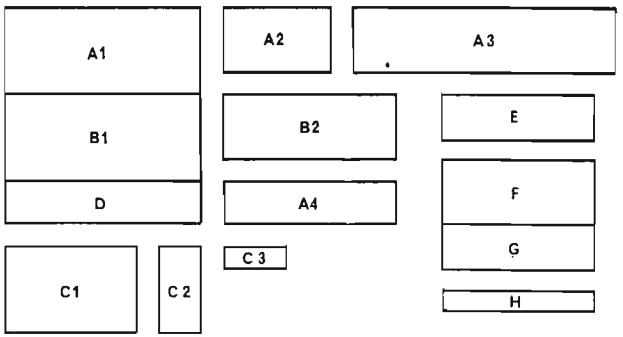


Figura A.1 - Mappa istruzioni Z80.

Istruzioni di trasferimento

Tabella A1: trasferimenti tra registri, immediati e indiretti.
La sorgente è specificata sull'intestazione delle colonne, la destinazione sulle righe.

Esempio:

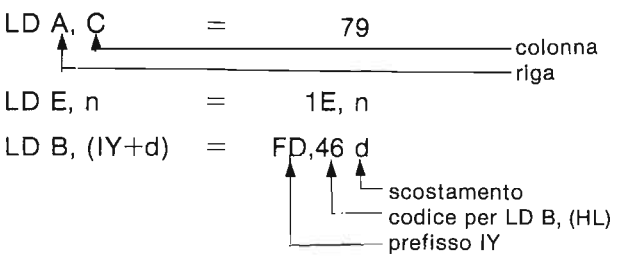


Tabella A2: trasferimenti con l'accumulatore A.
La freccia indica il verso del trasferimento
→ A: A è destinazione
← A: A è sorgente

Esempio:	LD (BC), A	= 02
	LD A, (nn)	= 3A, n,n

Tabella A3: trasferimenti su coppie di registri, PUSH e POP.
La freccia indica il verso del trasferimento, analogamente alla tabella A2.

Esempi:	LD (nn), DE	= ED,53, n,n
	LD IX, nn	= DD,21 n,n
	PUSH HL	= E5

Tabella A4: istruzioni di scambio.

Esempi:	EX AF, AF'	= 08
	EXX	= 09
	EX IY, (SP)	= FD,E3

Istruzioni aritmetiche e logiche

Tabella B1: operazioni su 8 bit
Le righe specificano il codice mnemonico della operazione, le colonne l'operando.

Esempi:	ADD A, (HL)	= 86
	CP n	= FE, n

Tabella B2: operazioni su 16 bit.
Righe e colonne come per la tabella B1; i codici operativi preceduti da un punto richiedono il prefisso ED.

Esempi:	ADD HL, DE	= 19
	SBC HL, BC	= ED,42
	ADD IX, BC	= DD,09

Istruzioni di salto

Tabella C1: salti e chiamate di sottoprogrammi.
Le colonne specificano la condizione di salto, le righe il tipo di istruzione.

Esempi:	JP 3200H	= C3,0032
	CALL NZ 8231H	= C4,31,82
	JR NC + 10	= 30,0A

Tabella C2: istruzioni di restart.
Tabella C3: altre istruzioni di salto.

Istruzioni di trasferimento sui periferici

Tabella D: L'uso è analogo alle istruzioni aritmetiche. Ricordare che, per le istruzioni di I/O con prefisso, l'indirizzo del periferico è nel registro C.

Esempi:	IN A, (F3H)	= DB,F3
	OUT (C), D	= ED,51

Istruzioni su blocchi di dati

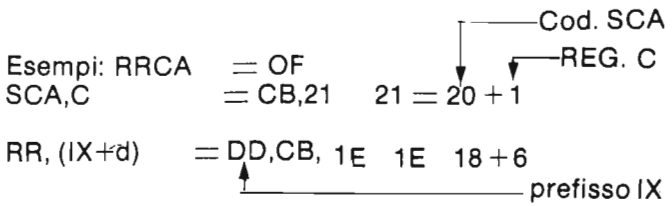
Tabella E: LoaD, ComPare, INput, OUTput.
Tutte queste istruzioni hanno il prefisso ED. Le indicazioni a lato richiamano le funzioni svolte.

Esempi:	I DIR	= ED,B0
	INR	= ED,AA

Istruzioni di rotazione e scalamento

Tabella F: Le istruzioni presenti nella prima colonna operano sull'accumulatore A. Quelle indicate nella seconda hanno prefisso CB, operano su tutti i registri ed il codice operativo completo si ottiene sommando (in esadecimale) la specificazione del registro (tabella H). È stata indicata anche l'istruzione SLI descritta nell'articolo citato.

Esempi:

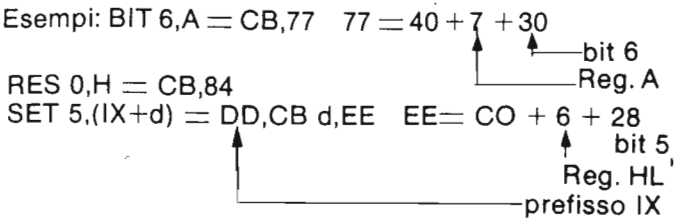


Istruzioni di manipolazione del bit

Tabella G: BIT, RESet, SET

Anche queste istruzioni hanno prefisso CB ed il codice si ottiene sommando identificazione del registro e del bit (tabella H fig. A-1).

Esempi:



Le altre istruzioni sono indicate singolarmente essendo di uso diretto.

APPENDICE B

LO STANDARD MUBUS

Questo standard hardware definisce un bus per sistemi a microprocessore indipendente dal tipo di unità centrale utilizzato.

I segnali del bus sono tali da permettere di interfacciare molto semplicemente memorie e controlli di periferici di ogni tipo.

Lo standard è nato dal confluire di iniziative di unificazione sorte in diversi laboratori universitari ed industriali, fin dal 1975 (vedi in proposito EUROMICRO NEWSLETTER n° 3, 1975 e sgg.). Le specifiche complete del MUBUS sono riportate in "MUBUS STANDARD" Microscope special issue, n°8/77 (editore J.D. Nicoud, p.o.box 141 CH 1007 - Lausanne).

Una descrizione del bus MMS8, che ha solo lievi modifiche rispetto al MUBUS, è comparsa in Elettronica oggi, n° 2,1979, col titolo "Una proposta di BUS standard per microcalcolatori modulari" di Bisani - Mezzalira - Negrini.

Il MUBUS è attualmente usato correntemente in parecchie università: sistemi e moduli per uso didattico ed industriale sono prodotti da alcune ditte italiane ed estere.

Lo standard MUBUS comprende una definizione funzionale, che specifica quali comandi esistono e come devono essere usati, ed uno standard fisico (formato delle schede, connettore ecc.).

Le tabelle di questo inserto riportano in forma compatta la definizione delle varie linee e l'assegnazione delle connessioni per cartelle formato Europa singolo con connettore diretto 2 × 37. Altri connettori e formati di scheda sono descritti nei riferimenti citati.

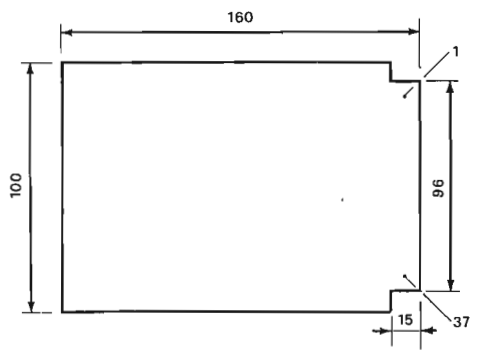


Figura B-1

Definizione funzionale delle linee di bus

Le linee sono raggruppate per classi di funzioni. L'asterisco indica una linea attiva a livello basso.

- Controllo dei trasferimenti dati:

ADD	(ADDress)	linee di indirizzo
DATA		linee dati
VMA*	(Valid Memory Address)	validazione di indirizzo per la memoria
VPA*	(Valid Peripheral Address)	validazione di indirizzo per un periferico
WR*	(WRite)	comando di scrittura
DEN	(Data ENable)	consenso al trasferimento dati
RDY	(ReaDy)	richiesta di attesa (WAIT)

- Comandi al master:

RST*	(ReSeT)	inizializzazione
INR*	(INterRupt	richiesta di interruzione
NMI*	(NonMaskable Interrupt)	richiesta di interruzione non mascherabile
HLDR*	(HoLD Request)	richiesta di sospensione (HOLD)
INH*	(INHibit)	isola il master dal bus

- Segnali di stato, abilitazioni a catena:

INA*	(INt. Acknowledge)	riconoscimento di interruzione
INI	(INt. In)	abilitazione a catena per l'interruzione (ingresso)
INO	(INt. Out)	abitazione a catena per l'interruzione (uscita)
HLDA	(HoLD Ackn.)	riconoscimento di HOLD
HLI	(Hold in)	abilitazione a catena per l'HOLD
HLO	(HoLd Out)	abilitazione a catena per l'HOLD
FTC*(1)	(FeTCh)	lettura del primo byte di una istruzione
ZPE*(1)	(Zero Page Enable)	abilitazione risorse locali della CPU in pagina 0
RFR*	(ReFResh)	ciclo di rinfresco per memorie dinamiche
SCK	(System Clock)	segnale di cadenza (2,4576 MHz)
LCK	(Line Clock)	segnale di cadenza (100 Hz)

+ 5
— 5
+ 12
— 12
GND
} **Alim. circuiti digitali**

(1) Questi segnali sono presenti nella sceda Picocomputer ma non sono definiti nello standard.

+ 15
— 15
GNDA } **Alim. circuiti analogici**

Connettore MUBUS

Connettore MUBUS per carte formato Europa (100 × 160 mm. conn. 2 × 37 diretto)

Pin n°	Lato componenti	Lato saldature
1	+ 15	+ 15
2	GNDA	GNDA
3	— 15	— 15
4	+ 12	+ 12
5	— 12	— 12
6	LCK	LCK
7	chiave di polarizzazione	
8	0	8
9	1	9
10	2	A
11	3	B
12	4	C
13	5	D
14	6	E
15	7	F
16	RFR*	SCK
17	NMI*	INH*
18	INR*	INA*
19	HLDR*	HLDA
20	RST*	DEN
21	WR*	VMA*
22	RDY	VPA*
23	0	FTC*
24	1	ZPE
25	2	
26	3	
27	4	
28	5	
29	6	
30	7	
31	INO	INI
32	HLO	HLI
33	— 5	— 5
34 35	+ 5	+ 5
36 37	GND	GND

ADD (pin 8–15, lato componenti)
ADD (pin 8–15, lato saldature)
DATA (pin 23–30)
Abitazione a catena (pin 31–32)

MUBUS - Diagrammi dei segnali fondamentali

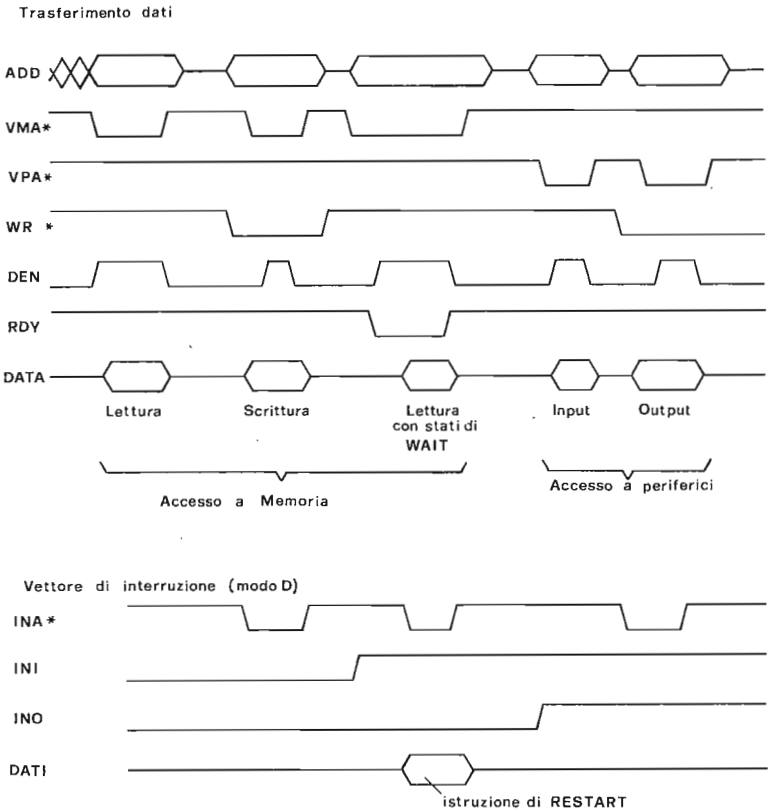


Figura B-2

Elenco delle schede MUBUS disponibili compatibili con il PICOCOMPUTER e relativi fornitori.

	Produttore
• CPU Z80, 1K ROM, 256 RAM 8IN, 8 OUT	RDT
• CPU Z80, 2K ROM, 1K RAM, 8IN, 8 OUT (PICOCOMPUTER)	MESA/GOMA A
• CPU RCA 1802	RDT
• EPROM 2716	RDT
• RAM 4K/16K statica	RDT
• I/O seriale (UART) e parallelo	RDT

• I/O mista (PIO, USART, CTC)	GOMA
• EPROM 4K (2708)	RDT
• EPROM/RAM 16K (2716/4118)	GOMA
• Interfaccia video grafica	GOMA
• Pannello di controllo hardware	RDT
• Piastra madre (BUS)	RDT/GOMA
• Programmatore di EPROM (2708/2716)	RDT/GOMA
• RAM 32 K dinamica	GOMA/RDT
• Interfaccia cassette con EPROM	RDT
• Controller floppy	RDT/GOMA

Schede con lo stesso connettore e lievi differenze in alcuni segnali:

PANEL (Losanna, CH)
famiglia di schede molto completa; compatibili per tutto tranne che per l'alimentazione analogica.

EMMECI (Milano)
famiglia di schede completa; compatibili con piccole modifiche.

Indirizzi:

GOMA	Via Valgioie, 1 TORINO
RDT	Rad - Tronic - Corso Tassoni, 69 TORINO
MESA	Via Monterosa, 13 MILANO
PANEL	CH 1028 PREVERENGES (SVIZZERA)
EMMECI	Viale Stelvio, 21 MILANO

APPENDICE C

TASTIERA E DISPLAY TECNICHE DI INTERFACCIAMENTO

Elemento indispensabile per rendere operativo qualunque sistema a microprocessore è un organo per scambiare informazioni col mondo esterno, in altre parole un periferico. Nel caso di periferici classici (teletype, terminale video), il loro costo ha un incidenza significativa sul costo del sistema, arrivando al limite ai "single board computer" del costo di qualche centinaio di migliaia di lire, che richiedono qualche milione di lire di periferici. Questa diventa un grave ostacolo quando si vogliono organizzare molti posti di lavoro operanti su microelaboratori (caso tipico delle scuole) e per chi vuole accostarsi con approccio hobbystico al microprocessore, senza doversi impegnare in spese rilevanti.

Le funzioni minime richieste ad un periferico al fine di mettere l'interazione con l'operatore sono:
- possibilità di introdurre dati ed inviare comandi,
- presentazione dei dati,

e occorrono per questo, al minimo, un visualizzatore ed una tastiera.

Il Picocomputer utilizza proprio una interfaccia di operatore di questo tipo realizzando cosí un soddisfacente compromesso costo/prestazioni. Il visualizzatore a 7 segmenti permette di presentare facilmente caratteri esadecimali, ed è quindi adatto per rappresentare con due cifre parole di 8 bit. Insieme ad una tastiera con 24 tasti e con un opportuno programma di gestione si ottengono cosí con minima spesa tutte le funzioni di monitor: caricamento e lettura della memoria e di registri, avvio di programmi di utente, lettura e scrittura su cassette magnetiche, breakpoint, ecc.

Descrizione a blocchi

Il periferico comprende:
- visualizzatori,
- tastiera,
- circuiti di interfaccia.

Il visualizzatore consta di un certo numero di indicatori a 7 segmenti. Nel nostro caso sono stati scelti indicatori a LED (luce rossa), che sono direttamente interfacciabili con logiche TTL.

I catodi sono collegati insieme cifra per cifra, e gli anodi di segmento corrispondenti sono collegati tra le varie cifre, come in fig. C.1. Per far comparire una cifra in una determinata posizione si devono attivare contemporaneamente la selezione di cifra ed i segmenti opportuni. Un numero completo di piú cifre viene generato selezionando in successione le varie cifre ed inviando di volta in volta la codifica corrispondente sui segmenti (fig. C.2). Uno schema a blocchi che realizza queste funzioni è in fig. C.3.

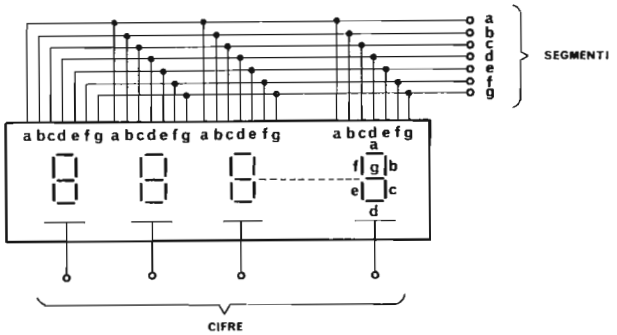


Figura C-1 - Visualizzatore 7 - segmeni multiplato

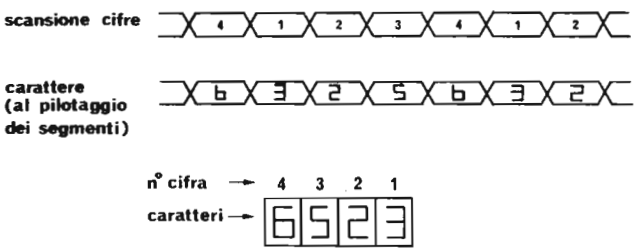


Figura C-2 - Comando di un visualizzatore multiplato a 4 cifre

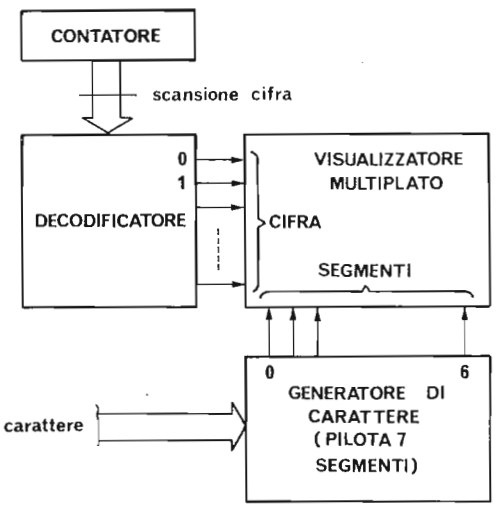


Figura C-3 - Pilotaggio di un display multiplato

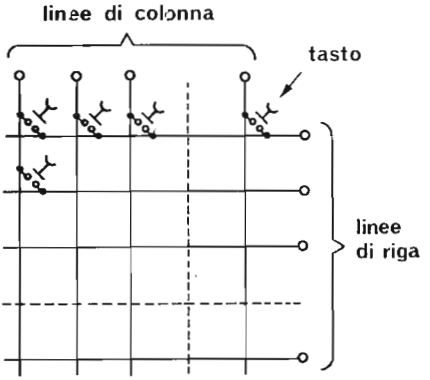


Figura C-4 - Tastiera a matrice

Noi vogliamo usare il visualizzatore per presentare dati elaborati dal microprocessore e quindi queste operazioni devono essere controllate da programma tramite le porte di I/O. Per la tastiera la configurazione piú comune è quella a matrice (fig. C.4). Premendo un tasto si connettono assieme una riga ed una colonna. Per generare il codice corrispondente a ciascun tasto possiamo impiegare lo schema funzionale della fig. C.5.

Supponiamo che le uscite del decodificatore e gli ingressi del codificatore siano attivi al livello basso (0), e che gli ingressi non collegati vadano al livello 1. Nella fig. C.5 solo una delle uscite COD (cioè un filo di colonna) per volta va allo 0 logico.

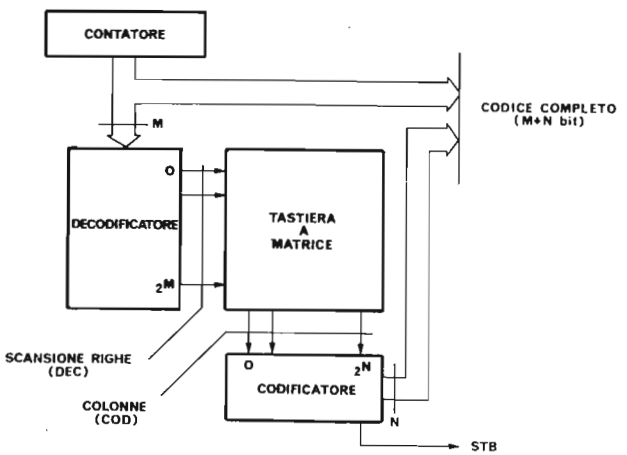
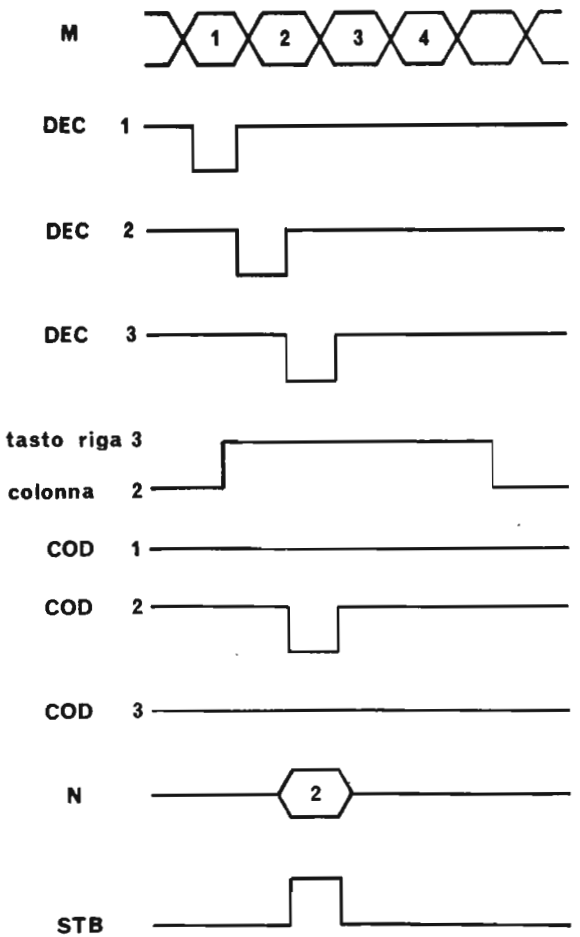


Figura C-5 - Scansione e codifica di una tastiera a matrice

Se ora viene premuto un tasto posto in questa colonna, lo 0 è riportato ad uno degli ingressi del codificatore che genera all'uscita una parola corrispondente al numero d'ordine dell'ingresso attivato ed un segnale STroBe, attivo ogni volta che uno qualsiasi degli ingressi va allo 0.STB può essere usato per indicare che un tasto è premuto (fig. C.6). Il codice che specifica il tasto si ottiene riunendo opportunamente l'uscita N del codificatore e l'ingresso M del decodificatore. Questo dato può essere memorizzato in un registro usando direttamente il segnale STB o, come nel nostro caso, letto dal microprocessore con una istruzione di INPUT.



Figuta C-6 - Segnali nel circuito di Figura C-4. DECI e CODI sono attivi allo ∅.

Interfacciamento

Gli schemi di fig. C.3 e C.5 sono schemi funzionali e la separazione tra operazioni eseguite a programma o implementate ad hardware può essere fatta a vari livelli . Ad esempio, nel caso del display, possiamo far pilotare direttamente dal microprocessore la selezione di cifra ed i segmenti (ovviamente tramite adatti buffer per fornire le correnti richieste) (Fig. C.7); in questo caso la scansione (contatore, decodifica) e la scelta dei segmenti da attivare per far comparire una determinata cifra (generatore di caratteri) sono realizzate a programma dal microprocessore.

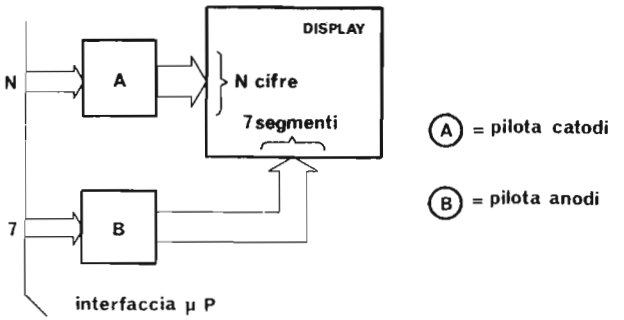


Figura C-7 - Comando diretto di un display a catodo comune

Un'altra soluzione può essere inviare la selezione codificata ed il codice del carattere, realizzando da hardware decodifica della cifra e dei segmenti (Fig. C.8). Ancora potremmo usare un contatore esterno a cui inviare solo i comandi di reset e clock, o adottare soluzioni miste tra quelle indicate. Nel primo caso un display a 7 segmenti di N cifre occupa N + 7 linee di uscita e si ha il vantaggio di poter formare qualunque carattere o simbolo, perchè ogni segmento può essere selezionato individualmente da programma.

La seconda soluzione, adottata nel Picocomputer permette di ridurre il numero di linee di interfaccia a $\log_2 N + 4$ (3 + 4 = 7 linee per un display di 8 cifre). Anche per la tastiera esistono possibilità analoghe, e scegliamo ancora di mantenere ad hardware i circuiti di decodifica-colonna/codifica-riga per limitare il numero di linee richiesto per

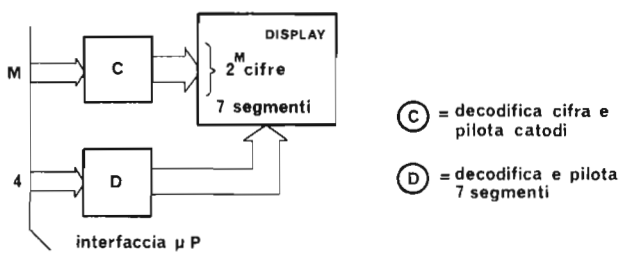


Figura C-8 - Comando codificato di un display (solo caratteri esadecimali)

l'interfacciamento. Sorge però un nuovo problema: la chiusura di un contatto meccanico (tasto) provoca una serie di rimbalzi che il microprocessore interpreterebbe come ripetizioni di una stessa operazione. Occorre quindi depurare le informazioni fornite dal codificatore di colonna (in particolare il segnale di STB) dei rimbalzi, ed anche questa è una operazione che può essere eseguita da hardware o da un programma.

La tecnica piú semplice per eliminare l'effetto dei rimbalzi è ricampionare la linea dopo un ritardo assegnato, tale da esaurire il transitorio a partire dalla prima transizione (Fig. C.9). Tastiera e visualizzatore richiedono entrambi una scansione, rispettivamente per indirizzare le cifre e le colonne. Nello schema proposto in seguito questa operazione è eseguita per entrambi dallo stesso circuito, riducendo cosí ancora le linee di interfaccia.

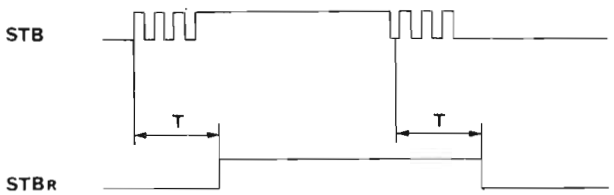


Figura C-9 - Rimbalzi sul segnale STB e segnale ricampionato STB_R

Programma di gestione

Per il visualizzatore le operazioni da compiere sono:
a) scansione delle cifre;
b) comando dei segmenti a seconda della informazione da visualizzare.

Dal momento che, come detto prima, decodifica della scansione e generatore di caratteri (esadecimali) sono implementati da hardware, le funzioni da gestire da programma sono:

- il contatore di scansione (SCAN);
- i trasferimenti da un'area di memoria che contiene i dati da visualizzare (BUFF).

Possiamo ora tracciare la sequenza delle operazioni (Fig. C.10). Da questo primo diagramma di flusso è possibile ricavare il programma di gestione completo con una serie di passi successivi, specificando via via i parametri non ancora definiti (formato sulla porta di uscita, formato dei dati in BUFF, ecc.). Occorre ancora definire il programma di gestione della tastiera, che, come è stato detto, utilizza lo stesso circuito di scansione del visualizzatore. È quindi comodo mantenere il programma di scansione del visualizzatore, ed inserire in questo un modulo per:

- verificare se è stato premuto un tasto,
- eseguire le operazioni di antirimbalzo,
- formare una parola corrispondente al tasto premuto.

Un primo schema a blocchi è nella figura C.11; il programma completo per la gestione di tastiera e visualizzatore è descritto in seguito.

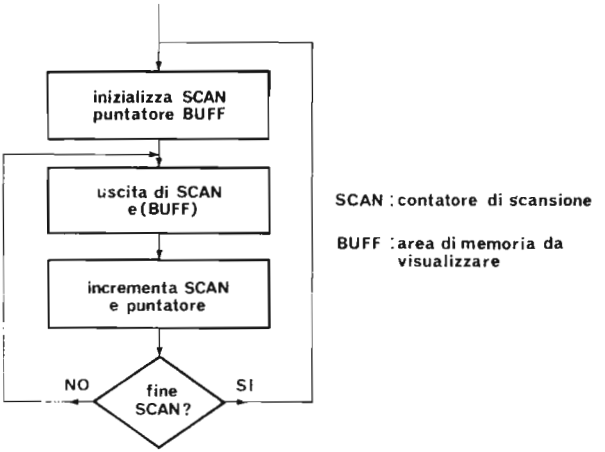


Figura C-10 - Operazioni da compiere per visualizzare sul display il contenuto di BUFF.

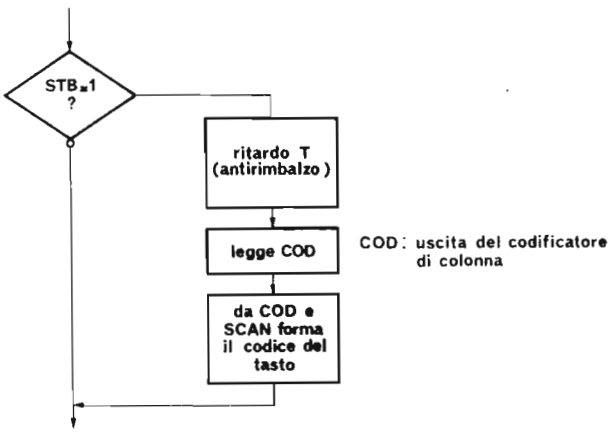


Figura C-11 - Modulo software per la gestione della tastiera a matrice.

Descrizione dell'hardware

Questo paragrafo descrive il circuito completo della scheda tastiera/display (Fig. C.12).

La tastiera comprende 6 righe e 4 colonne. Il codificatore 74LS148 è usato solo per metà (4 ingressi) e l'uscita COD è quindi solo di due bit. Dato che la scansione è fatta solo su otto linee (perchè il display ha 8 cifre e la tastiera 6 righe), l'ingresso D del deco-

dificatore 7445 nello schema di fig. C.12 va collegato a massa. Gli ingressi non utilizzati del codificatore 74LS148 vanno ad 1 attraverso le resistenze di pull-up.

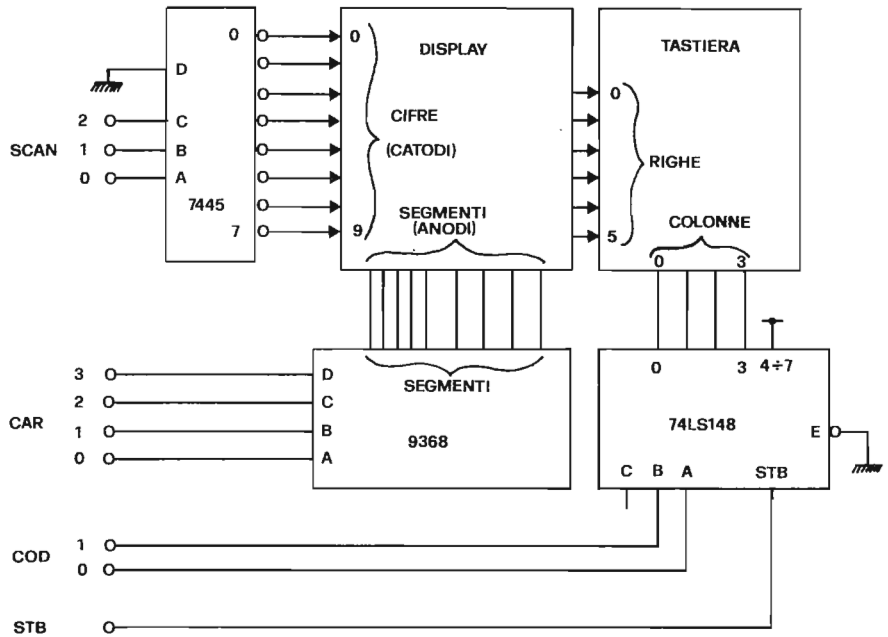


Figura C-12 - Schema elettrico

CAR 3 2 1 0	DECIMALE	ESADECIMALE (SUL DISPLAY)
1 0 0 1	9	9
1 0 1 0	10	A
1 0 1 1	11	b
1 1 0 0	12	C
1 1 0 1	13	d
1 1 1 0	14	E
1 1 1 1	15	F

Figura C-13 - Rappresentazione dei caratteri esadecimali usando il decodificatore 9368

Il programma presenta continuativamente sul display, in esadecimale, il contenuto di quattro celle di memoria, da BUFF a BUFF + 3, usando due caratteri per byte. Il contenuto di BUFF compare a partire dalle prime due cifre a destra (0 e 1). Quando viene premuto un tasto il programma forma nell'accumulatore A il codice corrispondente al tasto specifico. È poi possibile usare il contenuto di A come dato di ingresso o come comando al programma.

In PICOMON .V1 16 tasti sono usati per introdurre dati (in codice esadecimale), ed 8 per inviare comandi. Il programma comprende un loop principale (MAIN) che provvede alla scansione del display ed a prelevare da BUFF i dati da visualizzare. Un sottoprogramma PER prepara i dati nel formato corretto e verifica se è stato premuto un tasto (linea STB); in quest'ultimo caso viene eseguito un modulo che identifica dati e comandi (non riportato), e da questo ritorna al programma principale.

APPENDICE D

SCHEDA DI UNITÀ CENTRALE CRITERI DI PROGETTO E DESCRIZIONE DELL'HARDWARE

La scheda ha le dimensioni standard Europa (100 × 160 mm), comprende CPU Z80, RAM, ROM, interfacce di I/O e buffer di espansione secondo lo standard MUBUS (Appendice B). Dato che lo spazio disponibile è minimo, il progetto è stato particolarmente curato per ridurre il numero di componenti. Per lo stesso motivo non sono implementate alcune comode funzioni quali ad esempio la rilocalibilità della memoria locale, l'indirizzamento separato (I/O mapped) dei periferici locali, un restart automatico ad un monitor allocato a qualunque indirizzo, un real time clock e altro. Si è preferito invece lasciare sullo stampato dei posti integrato e zone millefori libere, disponibili per inserire altre funzioni e personalizzare cosí la scheda sulle necessità specifiche dell'utente. Un elenco dei possibili usi per questo spazio libero è in Appendice F.

Le dimensioni della RAM e ROM locali sono state scelte in modo da utilizzare componenti disponibili; si hanno quindi:

- fino a 2 Kbyte di ROM/EPROM, con possibilità di usare 2708, 2716 o PROM piú piccole;
- 1Kbyte di RAM (2 × 2114);
- due registri da 8 bit, mappati su memoria per utilizzare lo stesso decodificatore di indirizzo della ROM e RAM, utilizzabili per operazioni di I/O dall'esterno;
- due uscite impulsive, corrispondenti ad indirizzi non utilizzati sulla piastra, atte a comandare dispositivi esterni con memoria (contatori, registri,...).

Il microprocessore usato è lo Z80, dotato di un set di istruzioni potente ed esteso e soprattutto in grado di eseguire tutti i vari programmi già disponibili sul mercato per l'8080. In definitiva, la scheda comprende quindi, oltre al microprocessore con i relativi circuiti di controllo, anche memoria ed interfacce che permettono di utilizzarla sia da sola, come sistema minimo, sia come unità base di sistemi piú complessi.

Descrizione funzionale

I circuiti integrati MOS, quali microprocessori e memorie, hanno generalmente un Fan-out di 1 TTL standard, sufficiente per pilotare direttamente piccoli sistemi, ed in particolare tutte le risorse di una scheda singola di medie dimensioni. Per collegare piú moduli in vista di una effettiva espandibilità del sistema, occorre interporre sulle linee dei dati, indirizzo e controlli dei circuiti per aumentare la corrente disponibile (buffer) onde poter pilotare tutti gli ingressi collegati a ciascuna linea e le piste della piastra madre. Quindi, nel progettare una scheda che deve funzionare sia autonomamente che insieme ad altre, occorre decidere se bufferizzare o no le diverse linee di bus.

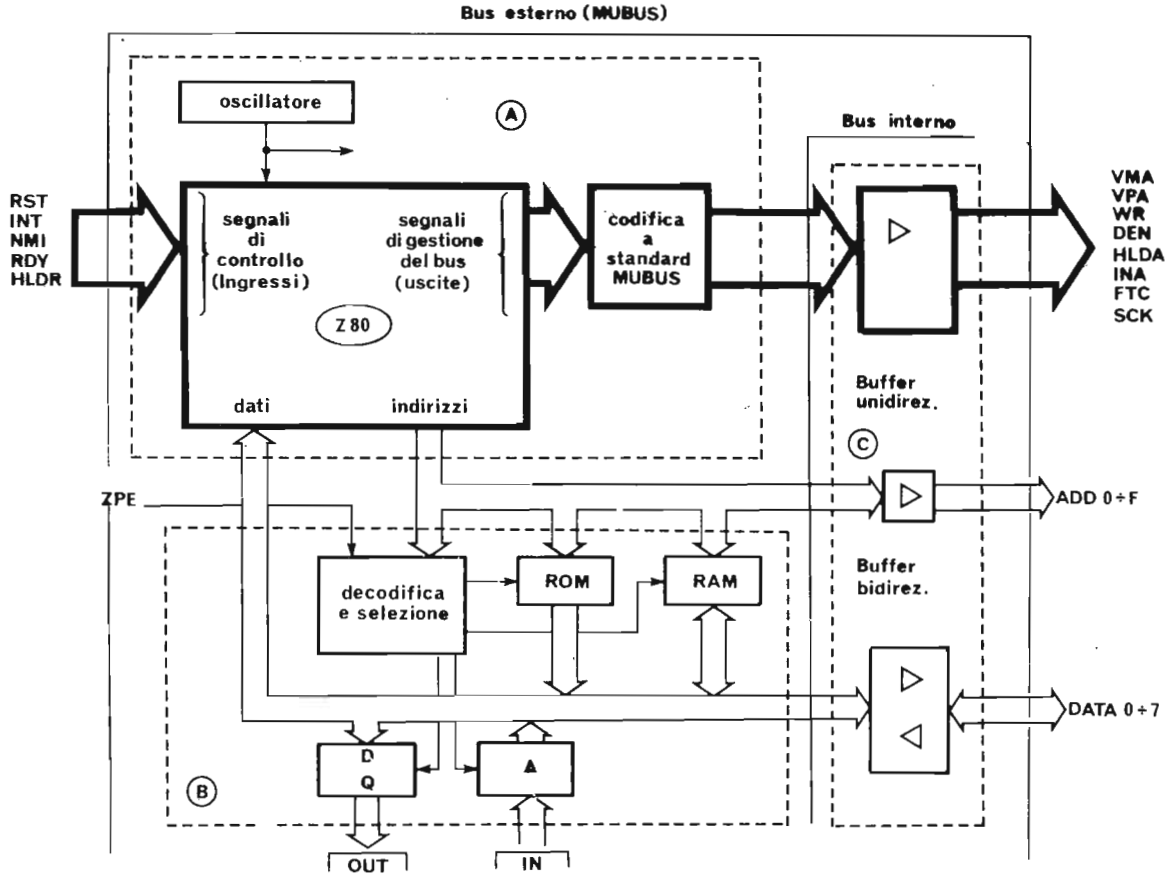


Figure D-1 - Schema a blocchi

La piastra CPU del PICOCOMPUTER è organizzata su due livelli di bus: bus interno, non bufferizzato, al quale sono collegate tutte le risorse locali quali RAM, ROM e interfacce, e bus esterno, connesso al resto del sistema. I due livelli sono collegati tramite buffer TTL. In questo modo la scheda può funzionare da sola in modo autonomo senza i buffer che devono essere inseriti solo in caso di espansione (i posti integrati sono già disponibili sullo stampato).

Lo schema a blocchi completo della scheda è in fig. D.1. In esso possiamo individuare tre moduli funzionali:

A - unità centrale di questione e controllo;
B - risorse locali (RAM, ROM, I/O) con i relativi circuiti di selezione;
C - buffer verso il bus esterno.

Il blocco A è il nucleo principale e può operare anche in assenza di B e C. Esso comprende:

- un oscillatore a quarzo,
- la CPU Z 80,
- i circuiti per la codifica dei segnali di gestione secondo lo standard MUBUS.

Il segnale dell'oscillatore viene inviato anche al resto del sistema (segnale SCK, linea 16 B) e la sua frequenza (2.4576 MHz) è quella standard dei generatori di cadenza per trasmissione seriale. È anche possibile usare frequenze diverse (fino a 4 MHz per lo Z80A); naturalmente, nel caso di frequenze piú alte occorre scegliere memorie con tempo di accesso adeguato. I segnali di controllo del bus Z80 sono trascodificati allo standard MUBUS attraverso un gruppo di porte e decodificatori.

Tutte le informazioni di stato (ciclo di rinfresco, riconoscimento dell'interruzione, accesso a memoria o periferico) sono decodificate ed inviate su linee separate del bus, definite nella tabella del MUBUS. (vedi fig. D.2). Viene anche generato un segnale di scrittura anticipato (WR), che semplifica l'interfacciamento al bus di qualsiasi tipo di memoria e può comandare direttamente la direzione dei buffer sulle linee dati. Il LED LD segnala l'esecuzione di una istruzione di HALT.

Le varie linee che portano comandi alla CPU (interruzione, reset ecc.), sono direttamente collegate ai rispettivi ingressi dello Z80, con una resistenza di pull-up per garantire il livello "1" logico (comando non attivo) a riposo.

Lo schema elettrico completo della scheda è in fig. D.5.

I diodi ed il condensatore C4 sulla linea di RESET sono necessari per generare un impulso di RST automaticamente all'accensione (C4 si carica lentamente verso il + 5V), mantenendo la possibilità di accettare impulsi di reset brevi (il diodo D 2 isola C4 carico dalla linea RST). Le risorse locali (blocco B in fig. D.1) sono tutte allocate in un banco di 4 Kbyte, a partire da pagina 0. La ROM inizia all'indirizzo 0000_H, e può essere usata per programmi di monitor o di inizializzazione, da attivare in seguito ad un reset. La mappa completa degli indirizzi è in fig D.3.

Il decodificatore di banco è un AND a 4 ingressi negati, che riconosce la configurazione 0000 sui bit di indirizzo ADD C ÷ F, realizzato con una connessione wired-and tra uscite e collettore aperto (invertitore 74LS05) . La selezione della memoria locale viene bloccata dal segnale ZPE (Zero Page Enable), che corrisponde alla linea 24B di bus e può essere controllato da un interruttore interno o esterno alla piastra, o da un bit di un periferico. Disabilitando le risorse locali della piastra CPU è possibile inserire memoria esterna, RAM o ROM, all'indirizzo 0000_H.

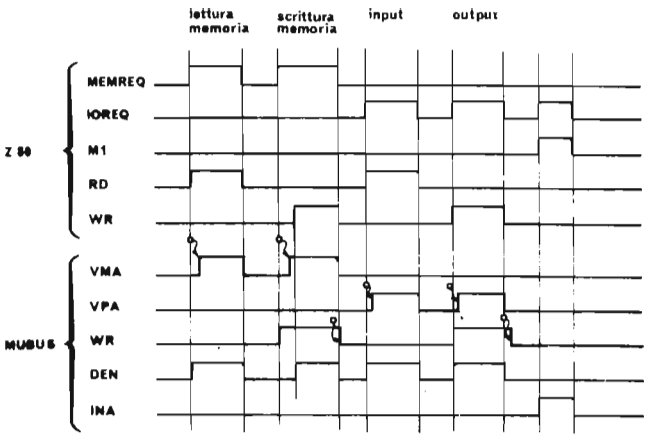


Figura D-2

Il decodificatore di banco genera il segnale OCA (On Card Address), che indica la selezione di una risorsa locale. Questo segnale viene usato come abilitazione (ingresso D) per un decodificatore 3-8 con uscite a collettore aperto (74LS145). La selezione segue la tabella di fig. D.3; le uscite che selezionano lo stesso dispositivo sono collegate in wired-or. I periferici sono mappati su indirizzi di memoria per evitare una logica di selezione separata; i registri di ingresso e di uscita corrispondono quindi ad un blocco di indirizzi di memoria, come specificato nella tabella. Eseguendo operazioni di scrittura sugli indirizzi assegnati alla ROM, questa non viene selezionata e si attivano in sua vece le uscite impulsive PUL 1 e PUL 2.

Tutte le linee di I/O sono portate su due zoccoli a 16 piedini (da usare con connettori per cavo piatto), secondo la tabella di fig. D.4. Ingressi ed uscite sono mescolati perchè il connettore A è previsto per il collegamento con la tastiera descritta nell'Appendice C.

Ingressi del decodificatore:	C	B	A		
linee di bus:	ADD B	ADD A	$\overline{WR}$	Risorsa	Indirizzo (Hex)
	0	0	0	PUL 1	0000 - 03FF
	0	0	1	ROM	0000 - 03FF
	0	1	0	PUL 2	0400 - 07FF
ADD F,E,D,C = 0	0	1	1	ROM	0400 - 07FF
$\overline{VMA} = 0$	1	0	0	RAM	0800 - 0BFF
$\overline{DEN} = 1$	1	0	1	RAM	0800 - 0BFF
$\overline{ZPE} = 0$	1	1	0	OUT	0C00 - 0FFF
	1	1	1	INP	0C00 - 0FFF

Figura D-3

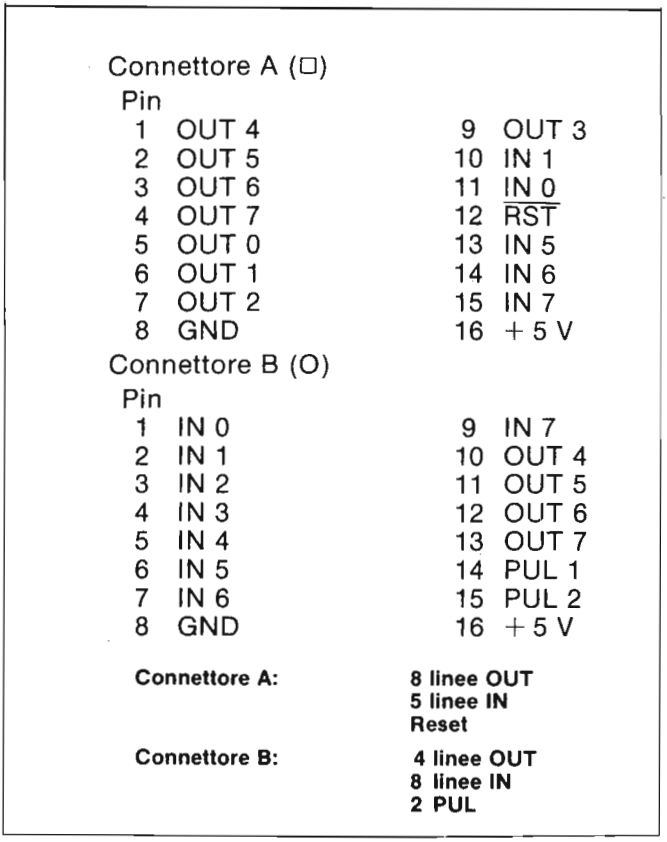


Figura D-4

I segnali PUL provengono dai transistori di uscita a collettore aperto del decodificatore 74LS145, in grado di assorbire 80mA e di reggere una tensione di 15 V.

Il blocco C comprende semplicemente dei buffer a tre stati che collegano il bus interno al bus esterno, accessibile al connettore della scheda. I buffer sono unidirezionali per le linee di indirizzo e di controllo, bidirezionali per le linee dati. L'interfacciamento tra i due livelli di bus mantiene la funzione di HOLD; questo comando porta in TRI-STATE tutte le uscite verso il bus ed il controllo può passare ad un altro master, per esempio un controllore di accesso diretto alla memoria. Esiste anche un comando INH (INHibit, linea 17 B), che isola il bus esterno da quello interno, indipendentemente dallo stato del processore.

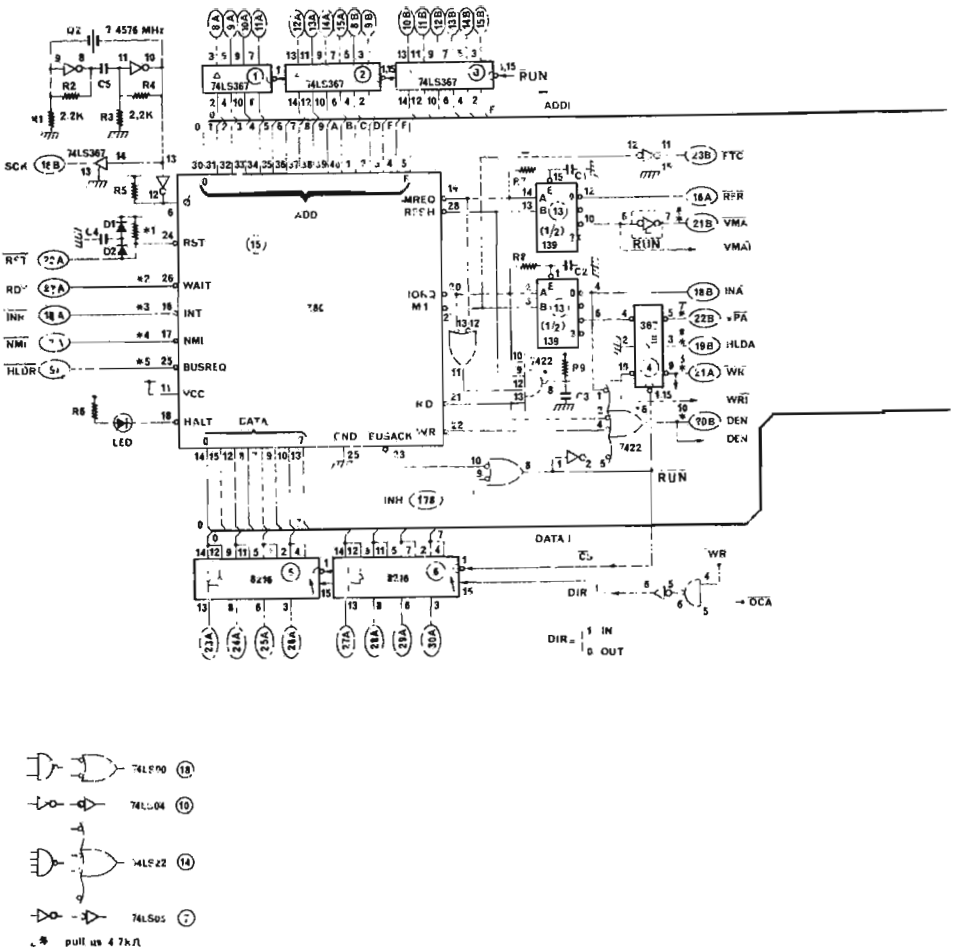
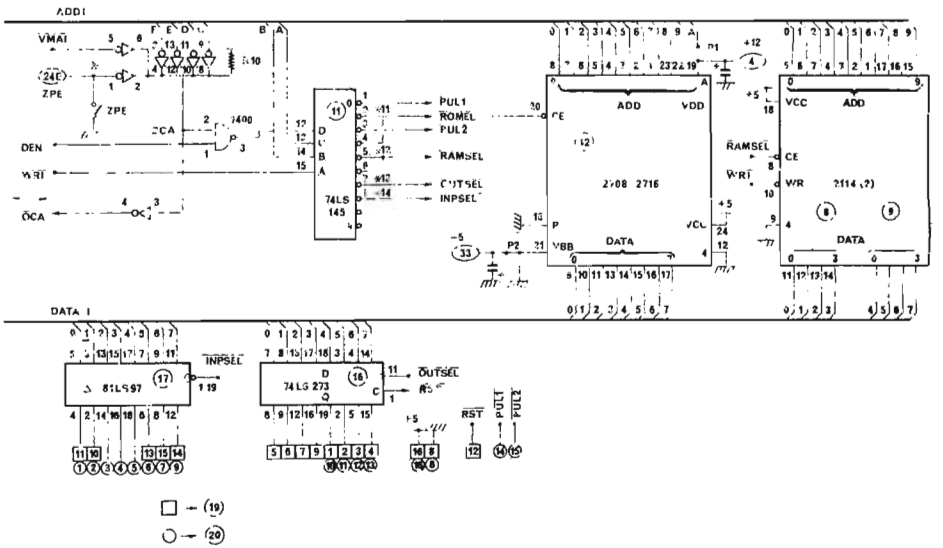


Figura D-5 - Schema elettrico completo


ADDI
VMAT
ZPE
DEN
WRT
ÖCA
74LS 145
PUL1
ROMEL
PUL2
RAMSEL
UTSEL
INPSEL
2708 2716
ADD
VDD
VBB
DATA
VCC
WR
2114 x2
DATA I
INPSEL
81LS97
74LS 273
ÖUTSEL
RST
PUL1
PUL2

APPENDICE E

SCHEDA DI UNITÀ CENTRALE: NOTE DI MONTAGGIO E COLLAUDO

Questa Appendice riporta alcuni suggerimenti per il collaudo della scheda CPU con strumenti standard (oscilloscopio e tester).

La scheda PICO può essere montata anche solo in parte, per esempio senza memoria ed I/O locale. I buffers n° 1, 2, 3, 4, 5, e 6 possono essere omessi se la scheda è usata da sola, o sostituiti con ponticelli per piccoli sistemi (fino a tre schede complessive).

A montaggio terminato è possibile eseguire alcune semplici operazioni di collaudo statico, che non richiedono l'uso di strumentazione sofisticata. Conviene per questo realizzare, anche solo su un bread-board, il circuito di Figura E.1. Con questo semplice "pannello di controllo", i comandi STOP e STEP permettono l'esecuzione passo passo di qualsiasi programma, ed il pulsante RESET invia il segnale di inizializzazione RST.

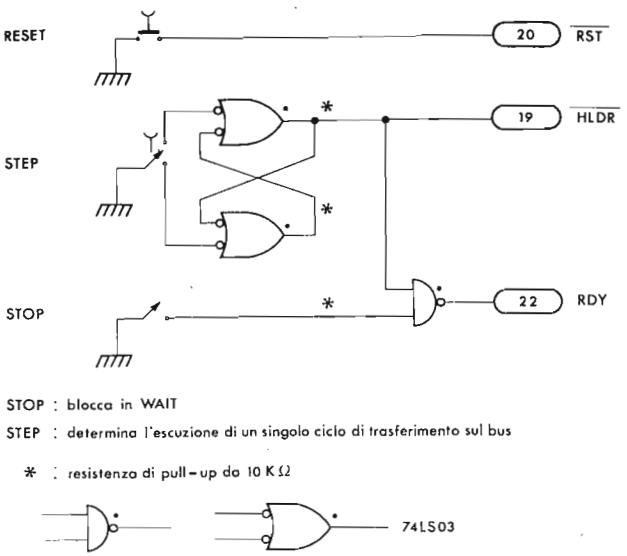


Figura E-1

Il funzionamento a ciclo singolo è basato sull'alternanza di stati di HOLD e WAIT; il microprocessore viene bloccato su ogni ciclo di accesso alla memoria o all'I/O dalla richiesta di WAIT (linea RDY = 0).

Premendo il pulsante STEP si porta RDY a 1, e si alza una richiesta di HOLD (linea $\overline{\text{HLDR}} = 0$); il ciclo di trasferimento così si conclude ed il processore va in HOLD. Rilasciando STEP la richiesta di HOLD viene rimossa e si riporta RDY a 0. Si inizia così, e si blocca immediatamente, il ciclo successivo della CPU. La sequenza temporale dei comandi è riportata nella Figura E.2. Questo circuito usa linee standard di Bus ($\overline{\text{HLDR}}$ e RDY) e funziona con qualsiasi microprocessore con funzioni HOLD e WAIT statiche.

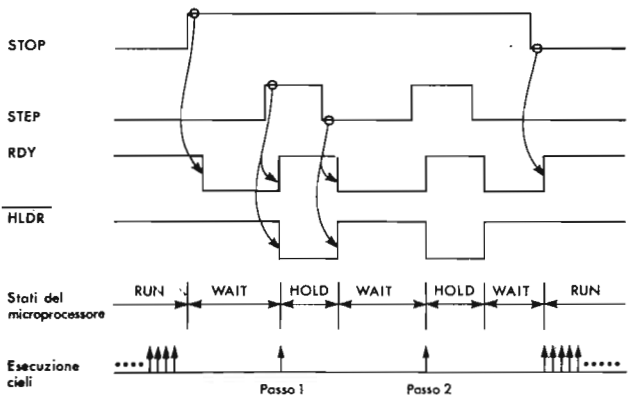


Figura E-2 - Esecuzione passo-passo: temporizzazione

Possiamo ora fare eseguire allo Z80 le varie operazioni (accessi a memoria ed I/O, salti, interruzioni ecc.), una alla volta, e verificare con semplici strumenti statici i diversi segnali, sia della CPU, sia sul bus esterno MUBUS (Figura E.3). Per visualizzare lo stato delle linee bastano dei LED, usando l'accortezza di interporre delle porte (ad esempio 74LS05) con funzione di buffer, per non caricare eccessivamente i punti in prova.

È anche possibile, e utile, realizzare un "pannello di monitoraggio" completo, che presenti contemporaneamente lo stato di tutte le linee di bus: indirizzi, dati e controlli.

Subito dopo aver collegato il circuito di passo-passo, è già possibile fare eseguire al picocomputer un programma, e precisamente una sequenza di NOP (No Operation). Infatti, dato che il codice operativo di questa istruzione è 00, basta collegare a massa le linee DATA 0-7 (conviene fare dei ponticelli sullo zoccolo della ROM, e disinserire RAM e buffer dati per evitare conflitti). L'istruzione NOP determina solo un incremento nel program counter; tenendo bloccato il microprocessore con il comando STOP, dopo un RESET tutte le linee di indirizzo vanno a 0 (Program Counter = 0000. Con un primo STEP avremo ADD 0 = 1 (PC = 0001), al successivo ADD 0 = 0 e ADD 1 = 1 (PC = 0002) e così via per 65.535 passi (fino a PC = $FFFF_H$).

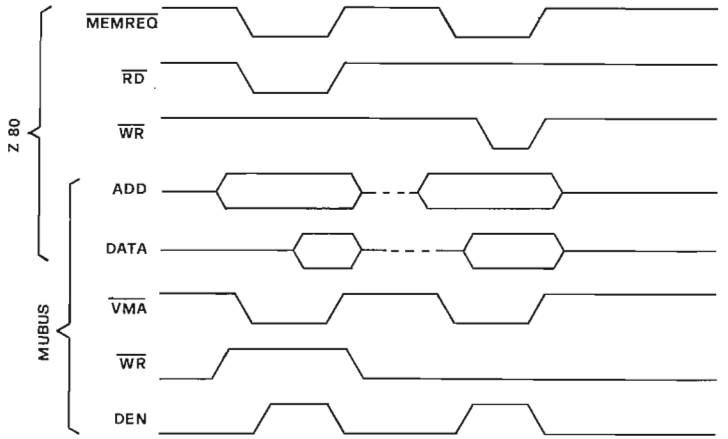


Figura E-3

Nell'esecuzione passo-passo i cicli di rinfresco vengono saltati perchè in essi lo Z80 non testa la linea WAITREQ (RDY nel nostro standard). Togliendo il comando STOP l'esecuzione procede alla cadenza normale ed è possibile osservare i vari segnali su un oscilloscopio. Questo metodo di collaudo a cicli singoli può essere esteso ad altre istruzioni; è sufficiente forzare sulle linee e dati (sempre attraverso i piedini della ROM) il codice operativo della istruzione che si vuol far eseguire. Conviene collegare un banco di 8 interruttori ove predisporre, un byte alla volta, il codice dell'istruzione, dare uno STEP, predisporre il byte successivo e così via.

In questo caso però le scritture che comportano cicli di scrittura o di output determinerebbero un conflitto di accesso sulle linee dati. E quindi necessario interporre tra bus ed interruttori, un buffer a 3 stati, come in Figura E.4. Il circuito di controllo abilita le uscite solo durante le operazioni di lettura (DEN = 1 e $\overline{WR}$ = 1).

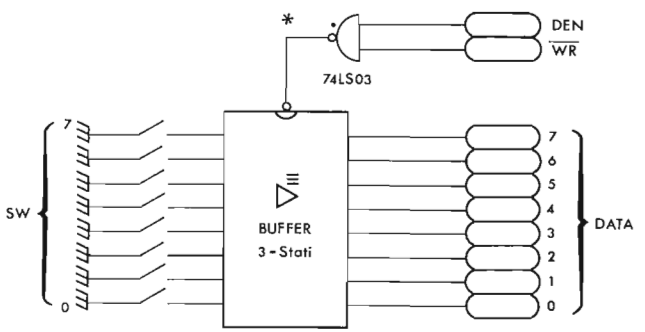


Figura E-4

A questo punto il Picocomputer può, con molta, molta pazienza, eseguire qualsiasi programma, servire interruzioni, fare in sostanza tutto, ovviamente un passo alla volta. Soffermarsi ad inventare e verificare esercizi e semplici programmi a questo livello è estremamente istruttivo e stimolante, ma anche scomodo e tutt'altro che rapido.

Conviene già dall'inizio avere a disposizione strumenti che permettano di preparare e far eseguire programmi in modo piú comodo e veloce. Utensile basilare per questo obiettivo è un complesso hardware (periferici) / software (programmi di debug, assemblatori, ecc.) tale da rendere piú semplici ed efficaci, e quindi meno soggetti ad errori, la preparazione ed il collaudo dei programmi di utente.

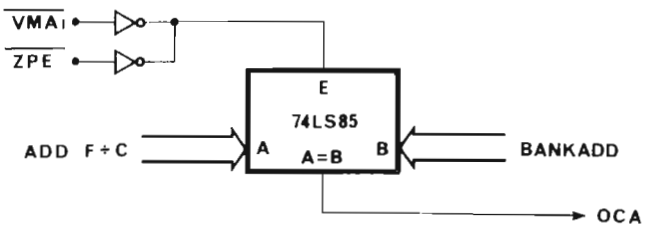
Questa generica definizione copre praticamente tutti gli "aiuti allo sviluppo", dai sistemi molto complessi che possono lavorare in linguaggi evoluti, fino al limite inferiore costituito dai microcalcolatori piú semplici, con periferici di basso costo (ad esempio tastiera e visualizzatore), supportati da programmi (detti monitor) che permettono solo di preparare e verificare i programmi di utente direttamente in linguaggio macchina.

APPENDICE F

SCHEDA DI UNITÁ CENTRALE: ESTENSIONI USANDO I POSTI INTEGRATO LIBERI

a) Rilocatore per le risorse locali.

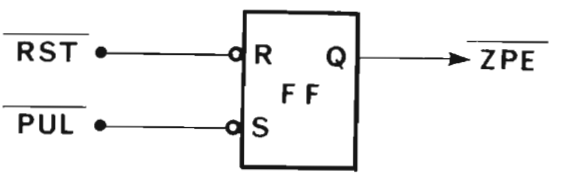
Sostituire il decodificatore di banco 0 (74LS05) con un comparatore (p. es. 74LS85).



È ora possibile assegnare la memoria e l'I/O locali ad un qualsiasi banco di 4Kbyte presettando opportunamente gli ingressi BANKADD.

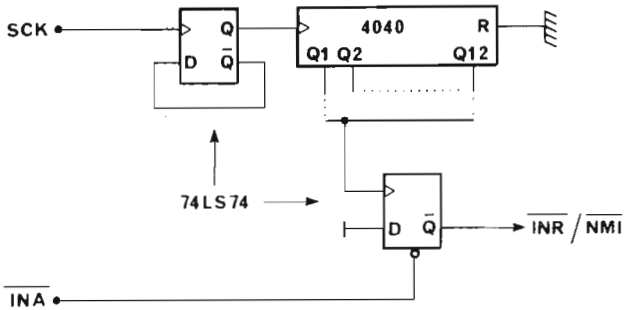
b) Bootstrap "fantasma":

la linea ZPE è controllata da un FF set-reset; dopo la esecuzione di un programma di inizializzazione contenuto nella ROM locale viene attivata un'uscita PUL che, attraverso il FF, disabilita le risorse locali ed abilita una RAM esterna da 4 K all'indirizzo 0000_H.



c) Real-time clock:

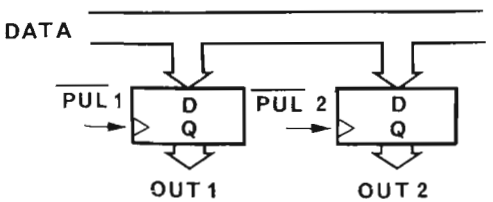
aggiungere un divisore ed un doppio FF tipo D.



La presa sul 4040 seleziona la cadenza delle interruzioni (fino a 3 ms circa). Non è prevista vettorizzazione; usare NMI o il modo 1.

d) Altre linee di uscita:

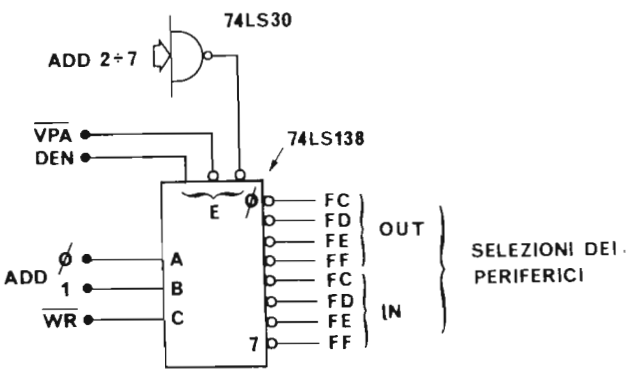
è sufficiente aggiungere registri strobati da PUL 1 e PUL 2:



Non è possibile aggiungere linee di ingresso perchè non sono piú disponibili indirizzi in lettura.

e) Mappamento su I/O dei periferici locali:

sostituire INPSEL o OUTSEL dei periferici locali con i segnali generati dal circuito qui indicato. Gli indirizzi selezionati vanno da FC a FF_H.



APPENDICE G

PROGRAMMA DI MONITOR ORGANIZZAZIONE ED ESTENSIONI

PICOMON .V1 (listato in Fig. G.1) è costituito da un programma principale, che visualizza sul display il contenuto di una zona di memoria (BUFF) e gestisce la tastiera, e dai sottoprogrammi di esecuzione dei vari comandi. Un diagramma di flusso semplificato è riportato in Figura G.2.

Durante l'inizializzazione viene predisposto (al valore di default $BD0_H$) anche il buffer dello stack di utente (cella SAVSP). Questo perchè l'esecuzione del comando di GOTO, che ripristina tutti i registri, richiede che lo stack di utente sia correttamente posizionato in RAM. Ciò è automaticamente garantito se si esce dal programma di utente con breakpoint (che salva tutti i registri, stack pointer compreso), ma può non essere vero per il primo GOTO dopo l'accensione del Picocomputer.

Il funzionamento del loop principale è quello descritto in VISTAST (BIT n° 3): premendo un tasto si genera nell'accumulatore il codice corrispondente alla combinazione riga/colonna del tasto premuto Figura G.3a. Dato che nella nostra scheda i registri di I/O sono mappati su memoria, le istruzioni di IN e OUT di VISTAST sono sostituite da istruzioni di LoaD (LD). La coppia di registri HL è usata come puntatore al periferico (indirizzo CONADD).

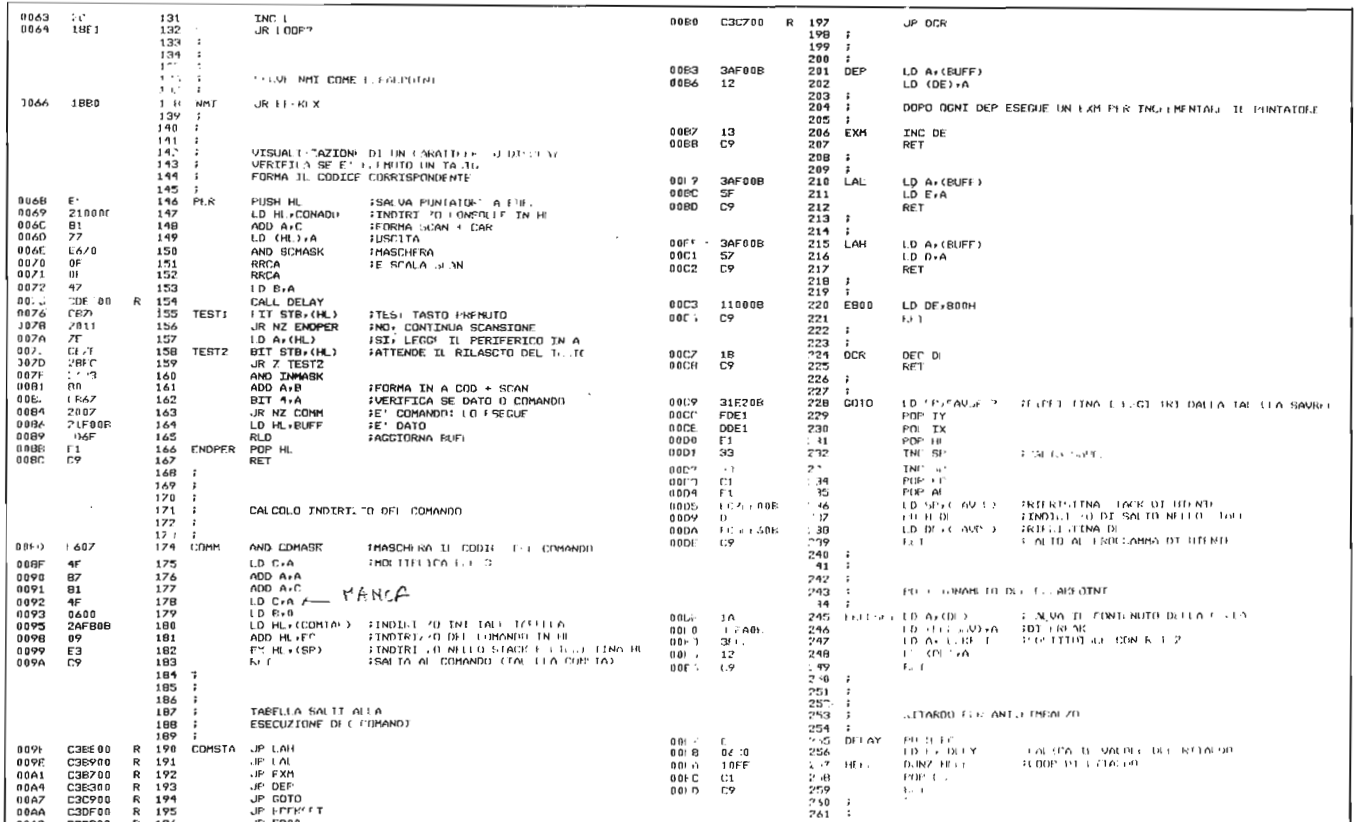
Mappa della tastiera e codici dei tasti sono in Figura G.3b. Una volta codificato il tasto, il programma determina se appartiene al campo dati (bit 4 = 0) o al campo comdandi (bit 4 = 1). I nuovi dati sono introdotti in un buffer temporaneo, presentato sulle due cifre piú a destra del visualizzatore; i comandi determinano un salto al corrispondente sottoprogramma di esecuzione. Questo salto ha luogo attraverso una tabella il cui indirizzo iniziale è definito in RAM (cella COMTAB), secondo la sequenza di operazioni di Figura G.4. È quindi possibile ridefinire le funzioni dei tasti nel campo comandi semplicemente alterando COMTAB e preparando un'altra tabella di salto, oltre ovviamente ai sottoprogrammi per l'esecuzione dei nuovi comandi.

Dopo l'esecuzione di un comando il controllo torna al monitor, tranne per il GOTO, che determina l'esecuzione del programma di utente. La gestione del visualizzatore e della tastiera è quella già descritta in Appendice C.

```
                                  PICOMONO        30.OTT 79          PAGE   1
LOC   OBJ CODE M STMT SOURCE STATEMENT                                ASM 5.8

                    1  ;
                    2  ;
                    3  ;
                    4  ;
                    5  ;        MONITOR [illegible]
                       ;
                    7  ;
                    8  ;
                    9  ;        DEFINIZIONE PARAMETRI
                   10  ;
                   11  ;
                   12  COMTAB  EQU    0BF8H    ;IND. INIZ. TABELLA COMANDI
                   13  SAVSP   EQU    0BF0H    ;SALVA SP
                   14  SAVDE   EQU    0BE6H    ;SALVA DE
                   15  SAVREG  EQU    0BE0H    ;AREA SALVATAGGIO REGISTRI
                   16  INTSER  EQU    0BFAH    ;ROUTINE DI INT SU BIT 7
                   17  BUFF    EQU    0BF0H    ;BUFFER DEL DISPLAY
                   18  UTSP    EQU    0BD0H    ;AREA STACK DI UTENTE
                   19  MOSP    EQU    0BE0H    ;AREA STACK DI MONITOR
                   20  BRKSAV  EQU    0BEAH    ;SALVA CELLA DI BREAK
                   21  MAXBUF  EQU    0F4H     ;LIMITE BUFFER DISPLAY
                   22  COMASK  EQU    60H      ;MASCHERA CONTATORE SCANSIONE [illegible]
                   23  LOMASK  EQU    0FH      ;MASCHERA NIBBLE BASSO
                   24  HIMASK  EQU    0F0H     ;MASCHERA NIBBLE ALTO
                   25  CONADD  EQU    0C00H    ;INDIRIZZO CONSOLLE
                   26  SCMASK  EQU    70H      ;MASCHERA CONTATORE SCANSIONE
                   27  INMASK  EQU    [illegible]H   ;MASCHERA [illegible] CODIFICA COLONNA TASTIERA
                   28  CDMASK  EQU    7        ;MASCHERA COMANDI
                   29  BRKPT   EQU    0DFH     ;[illegible] DA BREAKPOINT
                   30  DELY    EQU    80H      ;VALORE DEL RITARDO
                   31  STB     EQU    7        ;FLAG DI TASTO PREMUTO
                   32  ;
                   33  ;
                   34  ;
                   35  ;        INIZIALIZZAZIONI
                   36  ;
                   37  ;
0000  219B00   R   38  INIT    LD HL, COMTA     ;INDIRIZZO INIZIALE TABELLA COMANDI
0003  22F80B       39          LD (COMTAB), HL
0006  21D00B       40          LD HL,UTSP       ;INIZIALIZZA STACK DI UTENTE
0009  22C00B       41          LD (SAVSP),HL
000C  31E00B       42  INIT2   LD SP,MOSP       ;INIZIALIZZA STACK POINTER DEL MONITOR
000F  00           43          NOP              ;[illegible] INIZIALIZZAZIONI
0010  00           44          NOP
0011  00           45          NOP
                   46  ;
                   47  ;
                   48  ;
                   49  ;        PROGRAMMA PRINCIPALE
                   50  ;
                   51  ;
0012  CD3B00   R   52  MAIN    CALL COP         ;CHIAMA ROUTINE DI SCANSIONE E CODIFICA
0015  18FB         53          JR MAIN
0017  00           54          NOP
                   55  ;
                   56  ;
                   57  ;
                   58  ;        ESECUZIONE DI BREAKPOINT
                   59  ;        SALVA I REGISTRI DEL BANCO PRINCIPALE IN RAM
                   60  ;        PORTA IN DE L'INDIRIZZO DI BREAK
                   61  ;        RIPRISTINA IL CONTENUTO DELLA CELLA DI BREAK
                   62  ;
0018  ED53E60B     63  BREKEX  LD (SAVDE),DE    ;SALVA DE
001C  D1           64          POP DE           ;RICOSTRUISCE L'INDIRIZZO DEL BREAK
001D  1B           65          DEC DE
001E  ED73C00B     66          LD (SAVSP),SP    ;SALVA SP
0022  3AEA0B       67          LD A,(BRKSAV)    ;RIPRISTINA IL CONTENUTO DELLA CELLA
0025  12           68          LD (DE),A        ;DI BREAK
0026  31F00B       69          LD SP,SAVREG     ;INIZIALIZZA SP PER SALVATAGGIO REGISTRI
0029  D5           70          PUSH DE          ;SALVATAGGIO REGISTRI IN SAVREG
002A  F5           71          PUSH AF
002B  C5           72          PUSH BC
002C  3B           73          DEC SP           ;SALTA SAVDE
002D  3B           74          DEC SP
002E  E5           75          PUSH HL
002F  DDE5         76          PUSH IX
0031  FDE5         77          PUSH IY
0033  18D7         78          JR INIT2         ; AL MONITOR CON IND. DI BREAK IN DE
0035  00           79          NOP
0036  00           80          NOP
0037  00           81          NOP
                   82  ;
                   83  ;
                   84  ;
                   85  ;        SERVIZIO INTERRUPT SU BIT 7
                   86  ;
0038  C3[illegible]0B   87  INTR    JP INTSER        ;SALTA IN RAM
                   88  ;
                   89  ;
                   90  ;
                   91  ;        AGGIORNAMENTO DEL BUFFER DI DISPLAY
                   92  ;        COPIA:  D IN BUFF+3,
                   93  ;                E IN BUFF+2
                   94  ;                (DE) IN BUFF+1
                   95  ;
                   96  ;
003B  21F30B       97  COP     LD HL, BUFF+3
003E  72           98          LD (HL),D
003F  2B           99          DEC HL
0040  73          100          LD (HL),E
0041  2B          101          DEC HL
0042  1A          102          LD A,(DE)
0043  77          103          LD (HL),A
                  104  ;
                  105  ;
                  106  ;
                  107  ;        CANSIONE DISPLAY E TASTIERA
                  108  ;        QUESTO SOTTOPROGRAMMA
                  109  ;        CORRISPONDE AL MAIN DEL PROGRAMMA VISTO
                  110  ;
0044  21F00B      111  LOOP1   LD HL,BUFF
0047  7D          112  LOOP2   LD A,L
0048  FEF4        113          CP MAXBUF
004A  C8          114          RET Z
004B  0F          115          RRCA
004C  0F          116          RRCA
004D  0F          117          RRCA
004E  E660        118          AND COMASK
0050  4F          119          LD C,A
0051  7E          120          LD A,(HL)
0052  E60F        121          AND LOMASK
0054  CD6800   R  122          CALL PER
0057  7E          123          LD A,(HL)
0058  E6F0        124          AND HIMASK
005A  0F          125          RRCA
005B  0F          126          RRCA
005C  0F          127          RRCA
005D  0F          128          RRCA
005E  CBE7        129          SET 4,A
0060  CD6800   R  130          CALL PER
```

Figura G.1 - Diagramma di flusso di PICOMON, V1. (continua)



```
0063   0C          131          INC C
0064   18F1        132          JR LOOP?
                   133  ;
                   134  ;
                   135  ;
                   136  ;       GESTIONE NMI COME INTERRUPT
                   137  ;
0066   18B0        138  NMI     JR ERROR
                   139  ;
                   140  ;
                   141  ;
                   142  ;       VISUALIZZAZIONE DI UN CARATTERE SUL DISPLAY
                   143  ;       VERIFICA SE E' STATO PREMUTO UN TASTO
                   144  ;       FORMA IL CODICE CORRISPONDENTE
                   145  ;
0068   E5          146  PLR     PUSH HL             ;SALVA PUNTATORE A PILA
0069   21000C      147          LD HL,CONADD        ;INDIRIZZO CONSOLLE IN HL
006C   81          148          ADD A,C             ;FORMA SCAN + CAR
006D   77          149          LD (HL),A           ;USCITA
006E   E670        150          AND SCMASK          ;MASCHERA
0070   0F          151          RRCA                ;E SCALA A DX
0071   0F          152          RRCA
0072   47          153          LD B,A
0073   CDE800  R   154          CALL DELAY
0076   CB70        155  TEST1   BIT STB,(HL)       ;TEST TASTO PREMUTO
0078   2811        156          JR NZ ENDPER        ;NO: CONTINUA SCANSIONE
007A   7E          157          LD A,(HL)           ;SI; LEGGI IL PERIFERICO IN A
007B   CB7F        158  TEST2   BIT STB,(HL)       ;ATTENDE IL RILASCIO DEL TASTO
007D   28FC        159          JR Z TEST2
007F   E60F        160          AND INMASK
0081   80          161          ADD A,B             ;FORMA IN A COD + SCAN
0082   CB67        162          BIT 4,A             ;VERIFICA SE DATO O COMANDO
0084   2807        163          JR NZ COMM          ;E' COMANDO: LO ESEGUE
0086   21F80B      164          LD HL,BUFF          ;E' DATO
0089   ED6F        165          RLD                 ;AGGIORNA BUFF
008B   E1          166  ENDPER  POP HL
008C   C9          167          RET
                   168  ;
                   169  ;
                   170  ;
                   171  ;       CALCOLO INDIRIZZO DEL COMANDO
                   172  ;
                   173  ;
008D   E607        174  COMM    AND COMASK          ;MASCHERA IL CODICE DEL COMANDO
008F   4F          175          LD C,A              ;MODIFICA IL BC
0090   87          176          ADD A,A
0091   81          177          ADD A,C   <-- MANCA
0092   4F          178          LD C,A
0093   0600        179          LD B,0
0095   2AF80B      180          LD HL,(COMTAB)      ;INDIRIZZO INIZIALE TABELLA
0098   09          181          ADD HL,BC           ;INDIRIZZO DEL COMANDO IN HL
0099   E3          182          EX HL,(SP)          ;INDIRIZZO NELLO STACK E PILA IN HL
009A   C9          183          RET                 ;SALTO AL COMANDO (TRAMITE PILA)
                   184  ;
                   185  ;
                   186  ;
                   187  ;       TABELLA SALTI ALLA
                   188  ;       ESECUZIONE DEI COMANDI
                   189  ;
009B   C3E800  R   190  COMSTA  JP LAH
009E   C3B900  R   191          JP LAL
00A1   C3B700  R   192          JP EXM
00A4   C3B300  R   193          JP DEP
00A7   C3C900  R   194          JP GOTO
00AA   C3DF00  R   195          JP ERROFT
00AD   C3C300  R   196          JP FB00

00B0   C3C700  R   197          JP OCR
                   198  ;
                   199  ;
                   200  ;
00B3   3AF80B      201  DEP     LD A,(BUFF)
00B6   12          202          LD (DE),A
                   203  ;
                   204  ;       DOPO OGNI DEP ESEGUE UN EXM PER INCREMENTARE IL PUNTATORE
                   205  ;
00B7   13          206  EXM     INC DE
00B8   C9          207          RET
                   208  ;
                   209  ;
00B9   3AF80B      210  LAL     LD A,(BUFF)
00BC   5F          211          LD E,A
00BD   C9          212          RET
                   213  ;
                   214  ;
00BE   3AF80B      215  LAH     LD A,(BUFF)
00C1   57          216          LD D,A
00C2   C9          217          RET
                   218  ;
                   219  ;
00C3   11000B      220  EB00    LD DE,800H
00C6   C9          221          RET
                   222  ;
                   223  ;
00C7   1B          224  OCR     DEC DE
00C8   C9          225          RET
                   226  ;
                   227  ;
00C9   31E20B      228  GOTO    LD SP,(SAVSP)       ;RIPRISTINA LA PILA DALLA TABELLA SAVRET
00CC   FDE1        229          POP IY
00CE   DDE1        230          POP IX
00D0   E1          231          POP HL
00D1   33          232          INC SP              ;SCALA SUPER.
00D2   23          233          INC HL
00D3   E5          234          PUSH HL
00D4   F1          235          POP AF
00D5   ED7B0B00    236          LD SP,(SAVSP)       ;RIPRISTINA  IACK DI RITORNO
00D9   D5          237          PUSH DE             ;INDIRIZZO DI SALTO NELLO STACK
00DA   ED4B0B00    238          LD BC,(SAVBC)       ;RIPRISTINA BC
00DE   C9          239          RET                 ;SALTO AL PROGRAMMA DI UTENTE
                   240  ;
                   241  ;
                   242  ;
                   243  ;       PER I COMANDI DI DEBUGGING
                   244  ;
00DF   3A          245  ERROFT  LD A,(DE)          ;NUOVA IL CONTENUTO DELLA CELLA
00E0   12A0B       246          LD (BUFF),A         ;DI LAVORO
00E3   3EFF        247          LD A,0FFH           ;PREDISPOSIZIONE CONTATORE
00E5   12          248          LD (DE),A
00E6   C9          249          RET
                   250  ;
                   251  ;
                   252  ;
                   253  ;       RITARDO PER ANTIRIMBALZO
                   254  ;
00E8   E           255  DELAY   PUSH BC
00E8   0E00        256          LD C,DELAY          ;CARICA IL VALORE DEL RITARDO
00EB   10FE        257  HERE    DJNZ HERE           ;LOOP DI RITARDO
00ED   C1          258          POP BC
00EE   C9          259          RET
                   260  ;
                   261  ;
```

Figura G.1 - Diagramma di flusso di PICOMON, V1.

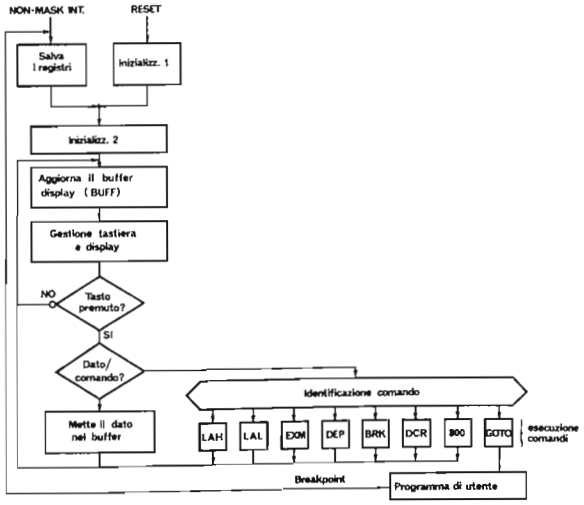


Figura G.2 - Diagramma di flusso di PICOMON. V1.

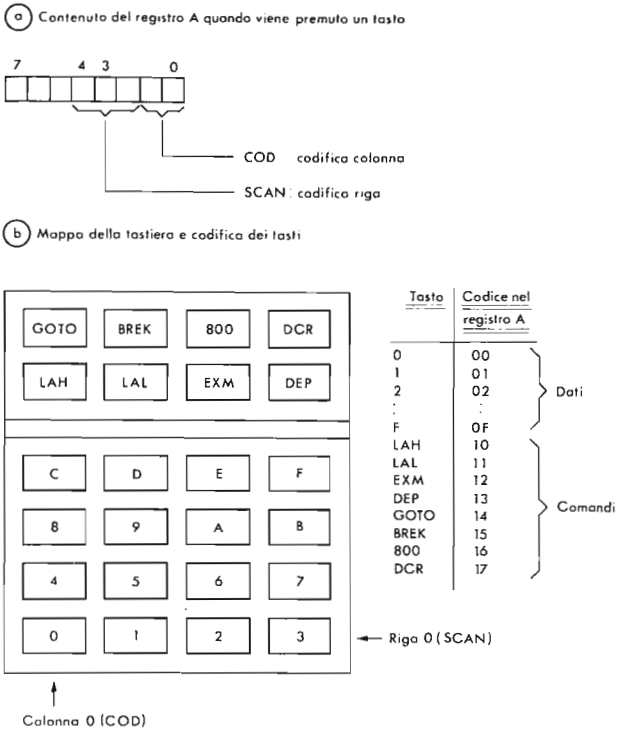


Figura G.3 - Mappa della tastiera e codifica dei tasti.

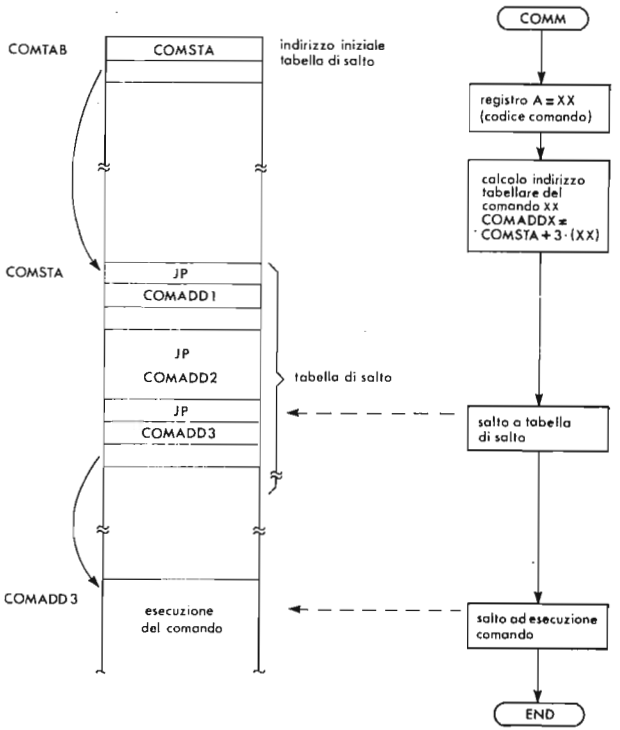


Figura G.4 - Salto indiretto all'esecuzione dei comandi.

APPENDICE H

INTERFACCIA CASSETTE MAGNETICHE

Le caratteristiche della interfaccia per cassette magnetiche del Picocomputer sono:

- uso di un normale registratore a cassette di media qualità;
- semplici circuiti per il modulatore e demodulatore;
- comando manuale del registratore;
- controllo di tutte le operazioni tramite la sola tastiera esadecimale.

In un sistema per registrazione e rilettura su cassetta possiamo individuare i blocchi funzionali di Figura H.1.

Con questa organizzazione ciascun modulo opera ad un ben definito livello ed è relativamente indipendente dagli altri. Ad esempio è possibile cambiare tipo di modulazione agendo solo sui blocchi C ed E, o modificare la struttura dei messaggi tramite i moduli A e G.

Come per ogni progetto che impiega il microprocessore, esistono diverse alternative per la ripartizione dei compiti tra hardware e software; le funzioni dei blocchi B ed F possono essere svolte da una interfaccia seriale tipo UART o USART, o dal programma, e lo stesso modulatore/demodulatore può essere realizzato almeno in parte a software. Nel nostro caso si è scelto di implementare a programma quante più funzioni possibile; solamente i blocchi C ed E sono realizzati con interfacce specifiche, per sfruttare al massimo le caratteristiche del canale.

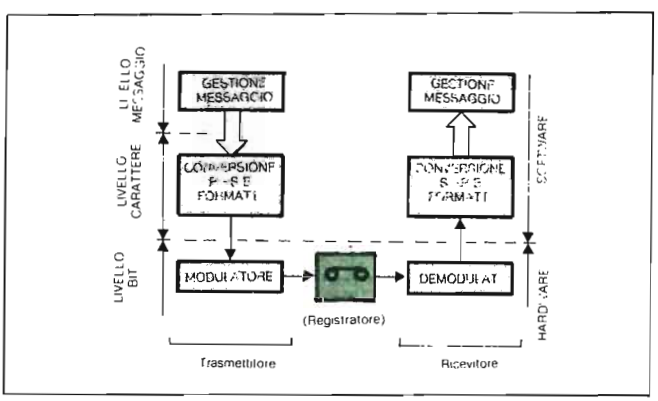


Figura H.1

Formato dei messaggi

I dati immagazzinati sulle cassette devono essere organizzati in messaggi che contengono tutte le informazioni necessarie per la fase di rilettura, cioè indirizzo a cui vanno scritti i dati, lunghezza del messaggio, controlli di errore.

Il formato scelto prevede l'invio diretto del contenuto delle celle di memoria su caratteri di 8 bit. La struttura dei messaggi è analoga a quella della banda perforata dei calcolatori PDP 11 DEC. Rispetto ad altri formati che inviano due caratteri ASCII per ogni byte di memoria (ad esempio il cosiddetto standard "industriale" Intel), si ha il vantaggio di dimezzare, a pari quantità di informazioni, la lunghezza del messaggio.

Il formato prevede blocchi dati per il trasferimento di informazioni e blocchi di "autostart" che specificano ove riprendere l'esecuzione al termine della lettura di un messaggio (v. Figura H.2a).

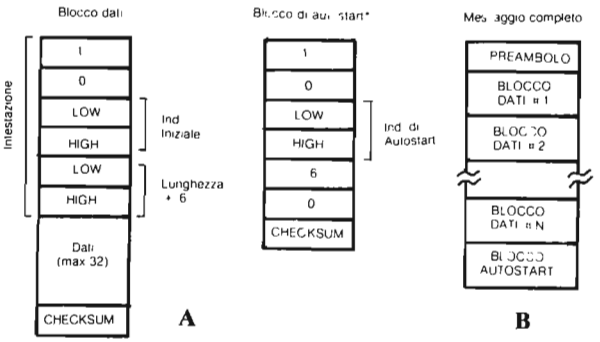


Figura H.2

In chiusura dev'essere sempre inviato un messaggio di autostart. La rivelazione di eventuali errori è affidata ad una parola di controllo (checksum), calcolata come somma modulo 2 bit a bit di tutti i caratteri del messaggio. Il carattere di checksum è trasmesso in coda a ciascun blocco. Per ridurre la probabilità di non segnalare errori multipli, la massima lunghezza di ciascun blocco è limitata a 32 bytes; messaggi più lunghi sono spezzati in una serie di blocchi consecutivi.

Ogni messaggio (v. Figura H.2b) comprende quindi:

- un preambolo, il cui scopo è chiarito in seguito:
- uno o più blocchi dati;
- un blocco di autostart.

Questo formato non prevede un'etichetta (cioè un nome) per i messaggi allo scopo di identificare i diversi programmi registrati, e quindi non è possibile in fase di lettura ricaricare solo un ben determinato programma, selezionato in base al nome, tra i molti registrati sulla stessa cassetta. Occorre portare il nastro in prossimità dell'inizio della registrazione che interessa (aiutandosi con il contametri e con commenti registrati a voce), e far partire la lettura da quel punto.

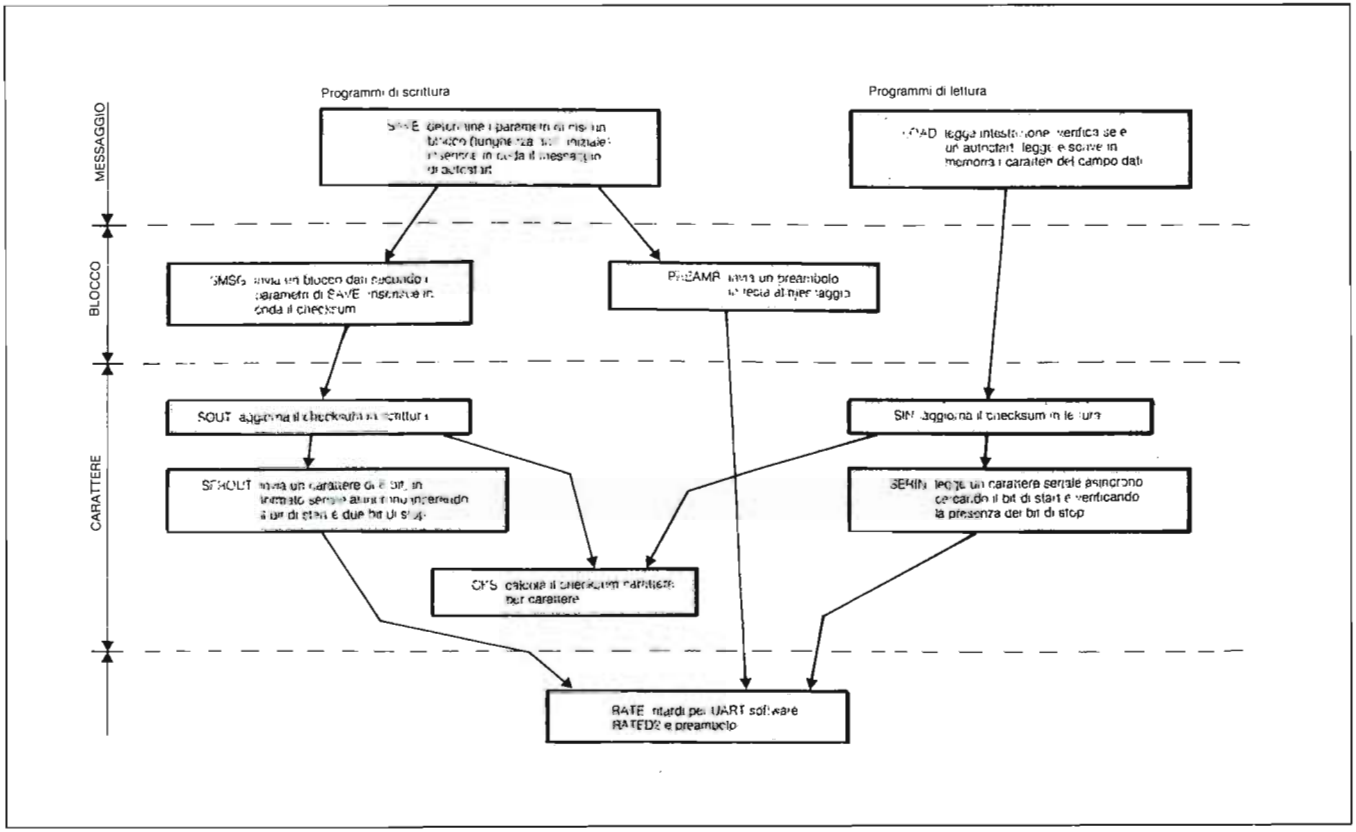

Programmi di scrittura
Programmi di lettura
MESSAGGIO
BLOCCO
CARATTERE
SAVE
LOAD legge intestazione verifica se è un autostart legge e scrive in memoria i caratteri del campo dati
SMSG
PREAMB
SOUT aggiorna il checksum in scrittura
SIN aggiorna il checksum in lettura
SPAOUT
SERIN legge un carattere seriale asincrono cercando il bit di start e verificando la presenza dei bit di stop
CHKS calcola il checksum carattere per carattere
RATE ritardi per UART software
RATED2 e preampolo

Figura H.3

Per svincolarsi dalle variazioni di velocità del registratore, senza dover ricorrere a complicati circuiti per la separazione dati/portante, ogni carattere viene inviato in modo asincrono, completato da un bit di start e due bit di stop. (Fig. H.4).

Programmi di gestione

Il complesso di funzioni da espletare per la gestione di messaggi è suddiviso tra diversi sottoprogrammi organizzati in modo gerarchico, secondo lo schema di Figura H.3. Ogni modulo viene chiamato da quelli di livello superiore ed usa a sua volta i sottoprogrammi di livello più basso.

SERIN e SEROUT costituiscono una vera e propria UART software; la cadenza di bit è specificata dal parametro DELY.

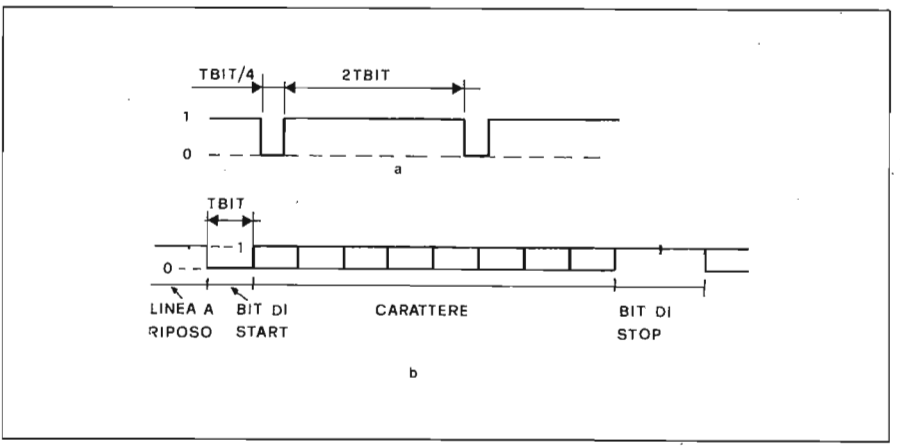


Figura H.4

Prima di ogni registrazione viene inviato un preambolo, costituito da una serie di brevi pacchetti della portante. Scopo del preambolo è far intervenire il controllo automatico di guadagno del registratore per lavorare poi a livello costante nella successiva fase dati. Durante la lettura il preambolo viene completamente ignorato dalla UART software. Questo perchè lo zero ha durata pari 1/4 T_{bit}, e quindi è sempre interpretato come disturbo e non come bit di start di un carattere asincrono (v. Figura H.4).

I diagrammi di flusso dei moduli software sono riportati nelle Figure H.5 e H.6.

I moduli principali SAVE e LOAD sono inseriti come comandi direttamente nel monitor stesso.

Prima di attivare SAVE occorre predisporre i parametri.

Il modulo di lettura LOAD, una volta attivato, termina solo per un blocco di autostart o per un errore. Ogni carattere letto sul nastro viene inviato sulla porta CONADD e fa quindi comparire un carattere sul display; questo permette di seguire il corretto svolgimento delle operazioni.

Durante l'operazione di LOAD vengono eseguiti due tipi di controllo di errore: a livello di singolo carattere si verifica la presenza di almeno un bit di stop, ed a livello di blocco viene controllato il checksum. In caso di errore termina l'operazione di lettura e viene segnalato, sul display del Picoperiferico, il codice di errore (sottoprogramma

ERR). Viene considerata errore e trattata come tale anche una intestazione diversa da quella standard (1,0). Questa uscita da LOAD può però essere usata anche per inserire blocchi dati particolari (ad esempio per l'identificazione dei programmi), individuati da una diversa intestazione. Dalla condizione di errore si esce solo con un RESET.

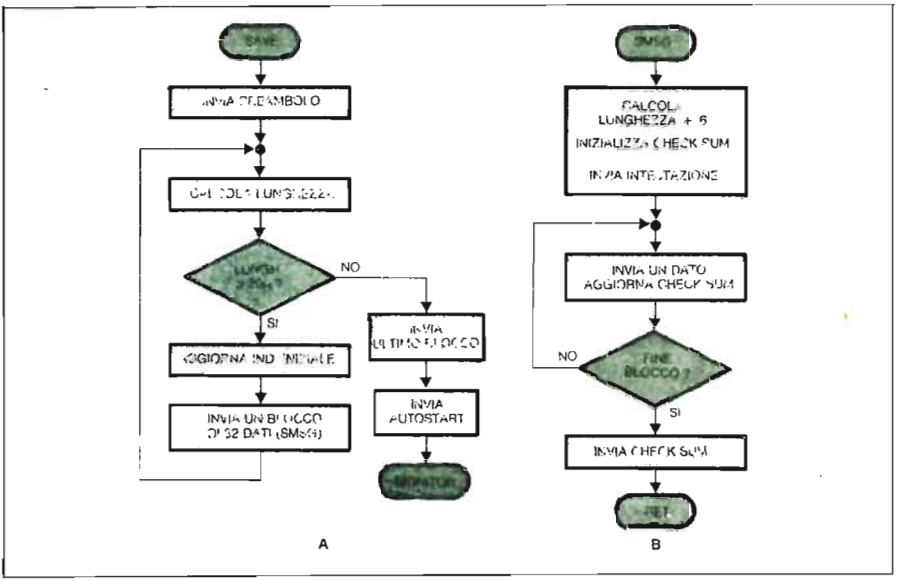


Figura H.5

Modulatore/demodulatore

Questo modulo è necessario per adattare la banda del segnale alla banda del canale (registratore), e concorre a determinare la massima densità di registrazione (bit/sec o bit/ cm) utilizzabile.

Una rassegna delle diverse tecniche di registrazione su supporto magnetico con indicazioni sulle prestazioni è contenuta nell'articolo: "Codice per la registrazione magnetica digitale...," di Graglia e Osella (Elettronica Oggi, n. 11 Novembre 1979).

Nel nostro caso, come compromesso tra semplicità, affidabilità e densità di registrazione, si è scelta la modulazione di ampiezza ON/OFF di una portante (AM al 100%). La densità è limitata principalmente dalle caratteristiche del registratore. Con il modulatore qui descritto si può arrivare a 1200 bit/sec; limitandosi a 600 bit/sec si ha una elevata affidabilità, ed il segnale può essere inviato su qualunque canale audio (anche su linea telefonica, attraverso accoppiatori acustici), con ottimi risultati.

Mantenendo gli stessi programmi di gestione, con opportune interfacce (vedi per esempio BIT n. 2) o collegandosi direttamente alle testine, è possibile arrivare, su registratori di buona qualità, a 9600 bit/sec.

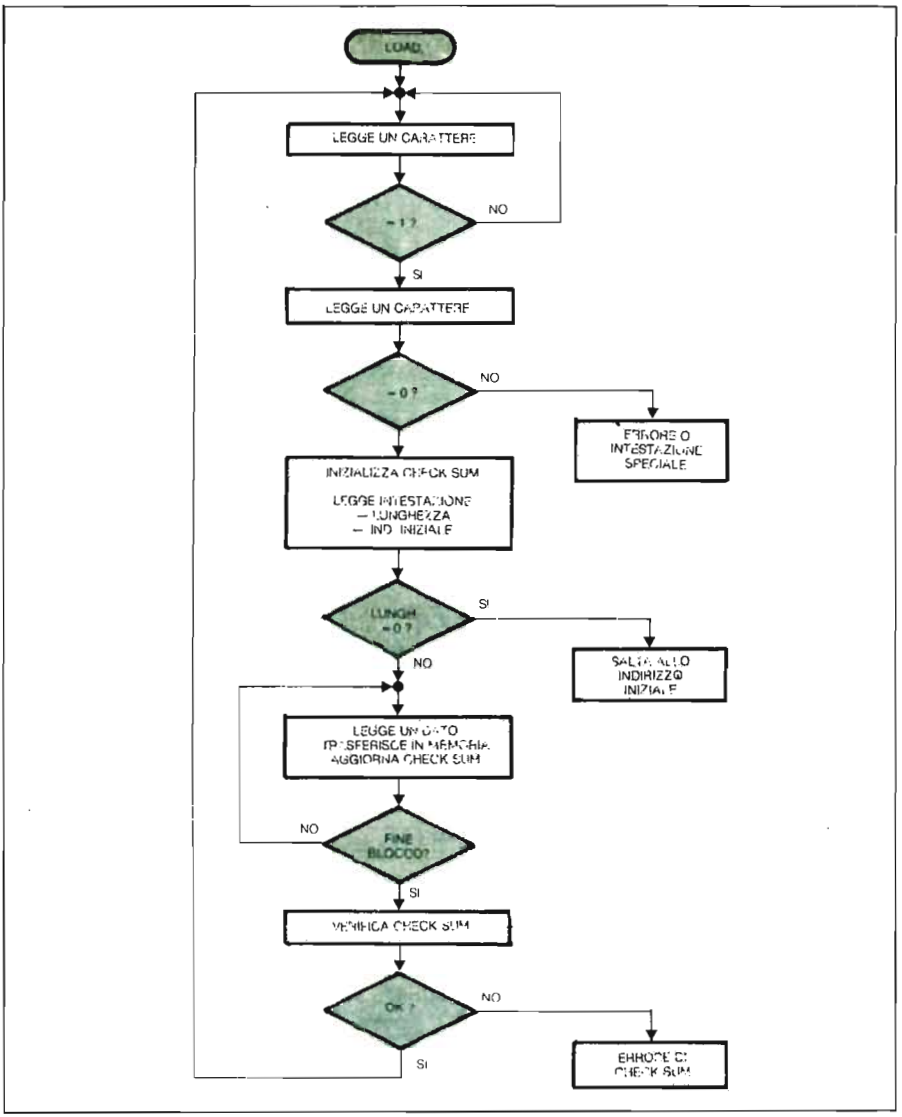


Figura 6 - Modulo per la lettura dei messaggi.

Figura H.6

Generare segnali PAM (Pulse Amplitude Modulation) non presenta particolari difficoltà; la sezione trasmittente dell'interfaccia è semplicemente un oscillatore controllato dal bit SO (Serial Out) della porta di uscita. Si ha emissione di nota in corrispondenza del livello 0; in questo modo le parti non registrate della cassetta sono interpretate come "linea a riposo" ed ignorate dall'UART (v. Figura H.4). Uno schema elettrico del modulatore è in Figura H.7.

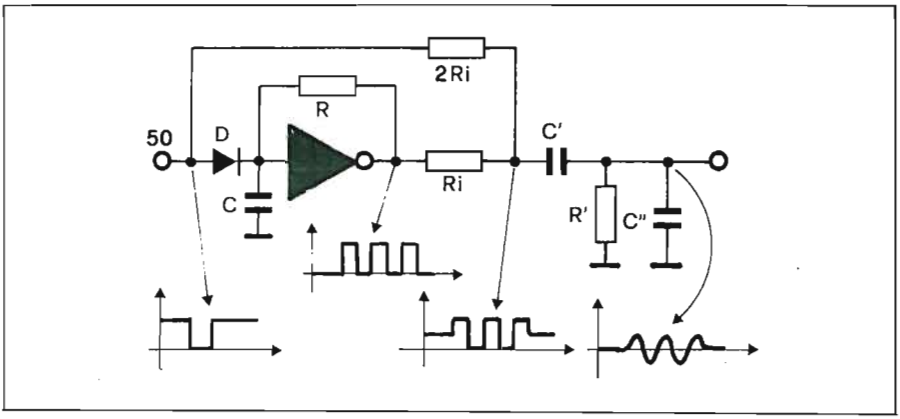


Figura H.7

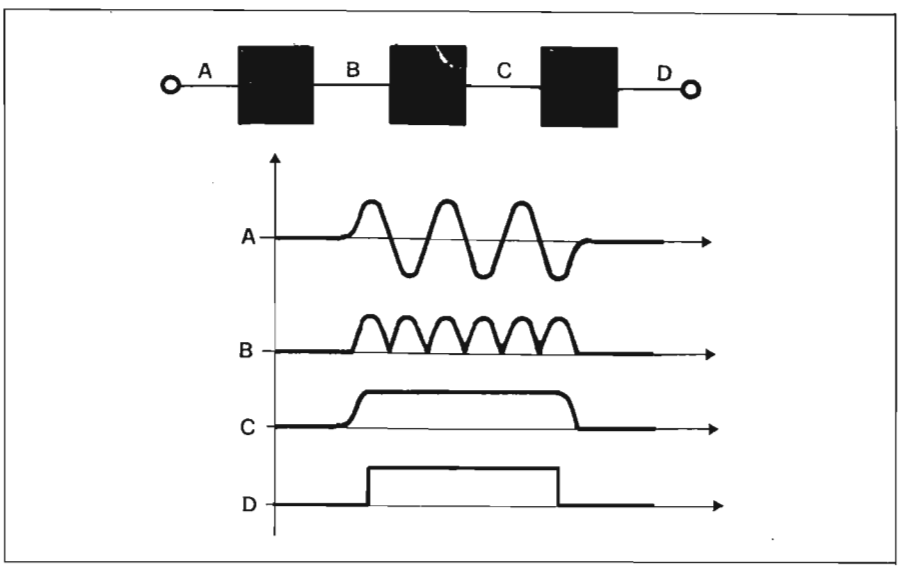


Figura H.8

L'oscillatore è un comparatore con isteresi reazionato RC. Il valore di questi due componenti determina la frequenza della portante. Il diodo D blocca l'oscillatore forzando ad 1 l'ingresso del comparatore. Le resistenze R1 e 2 R1 formano un sommatore che elimina la componente continua in uscita nei periodi di riposo. Il filtro R' C' C''

porta il segnale a valor medio nullo ed elimina le armoniche dell'onda quadra. Il valore di R' determina il livello del segnale in uscita.

Più complesso il problema del demodulatore, dal quale dipendono in massima parte le prestazioni del sistema. Occorre tener conto delle distorsioni introdotte dal canale (cioè dal registratore), quali rumore, limitazione della banda, variazioni di velocità e di livello.

Il demodulatore classico per segnali PAM si ottiene con un raddrizzatore a doppia semionda seguito da un filtro passa-basso ed un comparatore di soglia (v. Figura H.8).

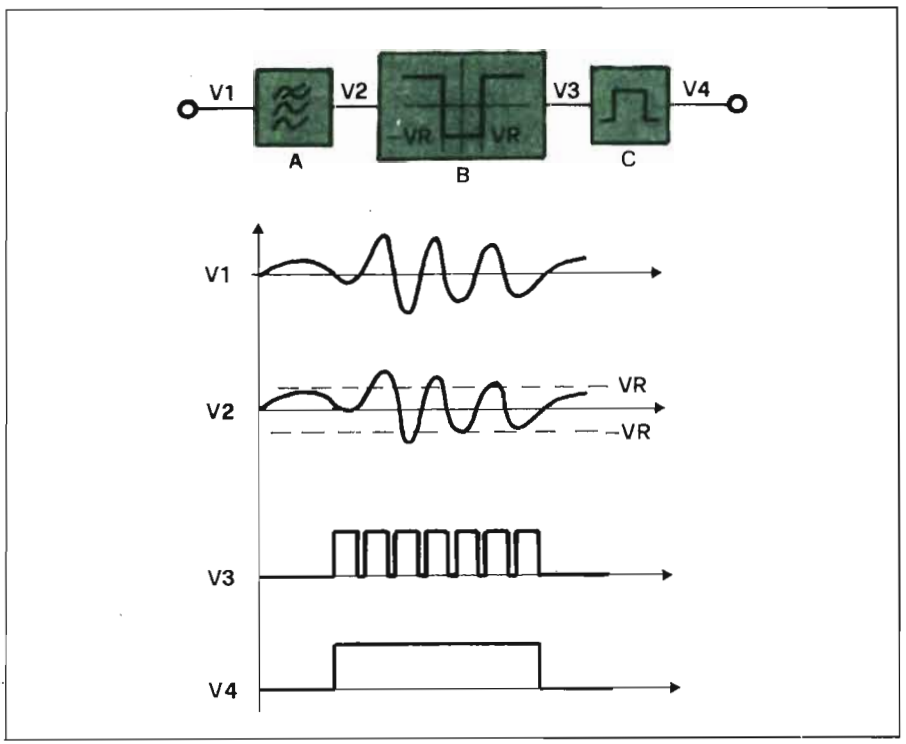


Figura H.9

Il circuito qui proposto è una variante del raddrizzatore ad onda intera; non usa diodi e può funzionare anche con una singola alimentazione a 5 V. Lo schema a blocchi è in Figura H.9. Il filtro passa-alto A elimina le fluttuazioni a bassa frequenza; il comparatore a finestra B rivela segnali di ampiezza superiore all'intervallo tra le soglie V_{R1} e V_{R2}. La sua uscita va all'allungatore di impulso D, che mantiene a livello alto l'uscita in presenza di impulsi ravvicinati provenienti dai comparatori. Uno schema elettrico

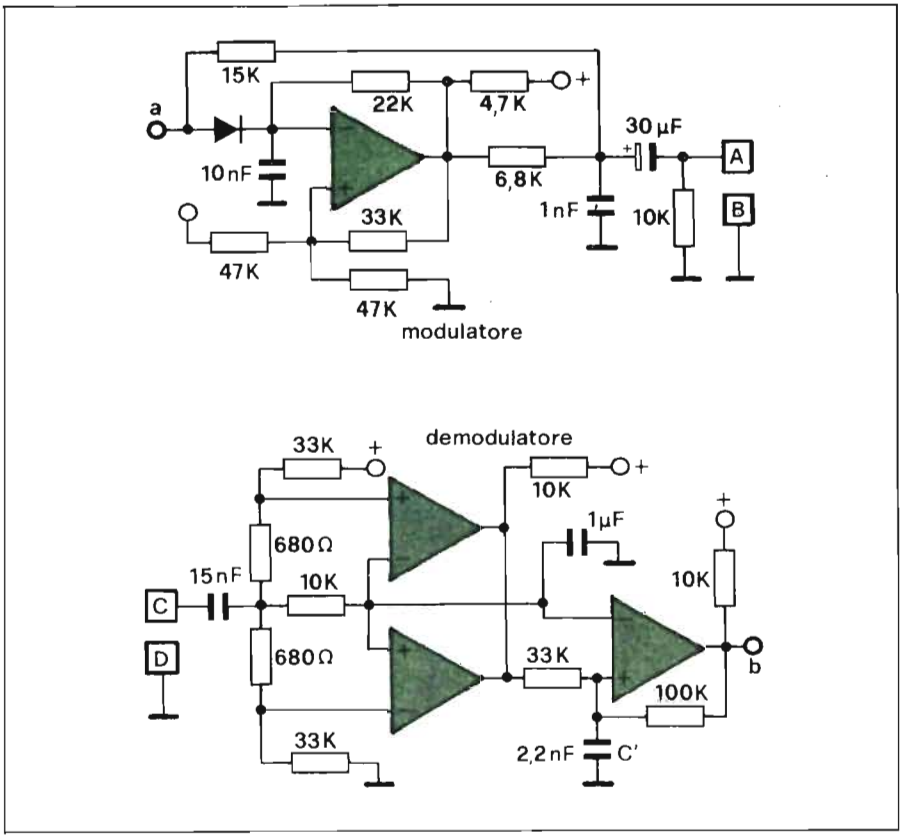


Figura H.10

completo è in Figura H.10. Rispetto allo schema funzionale si ha un'inversione logica, perchè la presenza di nota corrisponde allo 0. Il comparatore a finestra è scisso in due circuiti, rispettivamente per la soglia superiore e per la soglia inferiore, le cui uscite (del tipo collettore aperto) sono direttamente collegate in OR cablato.

Rallentando il tempo di salita con il condensatore C', si ottiene anche l'allungatore di impulso.

APPENDICE I

TECNICHE DI INTERFACCIAMENTO SU MUBUS

I moduli funzionali presenti in un microcalcoltore sono classificabili in due categorie:

MASTER: Sono i dispositivi che, di propria iniziativa, possono attivare una operazione di lettura o di scrittura verso la memoria o un periferico. Appartengono a queste categorie l'unità centrale ed i controllari di accesso diretto alla memoria (DMA).

SLAVE: Sono i dispositivi che partecipano alle operazioni di lettura e scrittura su comando di un master. Questa categoria comprende le memorie e le interfacce di periferici.

La scheda Picocomputer ospita un master (lo Z 80) ed alcuni slave (memoria RAM ed EPROM, registri di I/O).

Vedremo ora come si progettano altre schede contenenti moduli di tipo slave, necessarie per realizzare le espansioni di memoria e dei periferici del Picocomputer. Per semplicità prenderemo in esame solamente memorie statiche e periferici che non prevedono la generazione di interruzioni.

Dal momento che il Picocomputer è interfacciato secondo lo standard MUBUS (Appendice B), anche i moduli slave devono essere organizzati secondo le stesse linee di controllo.

Questo bus è stato definito in base alle esigenze "di sistema", e l'interfacciamento degli slave è particolarmente semplice e diretto.

Struttura di un modulo slave

In base alla definizione data in precedenza, qualsiasi modulo slave, sia di memoria che di periferico, deve:

a) riconoscere la propria attivazione, cioè decodificare il proprio indirizzo ed i segnali di controllo e validazione inviati dal Master;
b) eseguire l'operazione di trasferimento dati richiesta (lettura o scrittura).

I blocchi funzionali che lo compongono sono quindi (vedi fig. I.1):

1) decodificatore di indirizzo;
2) buffer dati e relativa logica di controllo;
3) "cuore" vero e proprio del modulo, cioè banchi di memoria o interfacce di I/O.

Lo stato dei segnali durante i cicli di lettura e scrittura è riportato in fig. I.2. I segnali VMA/VPA (Valid Memory/Peripheral Address), attivano la fase di selezione (indiriz-

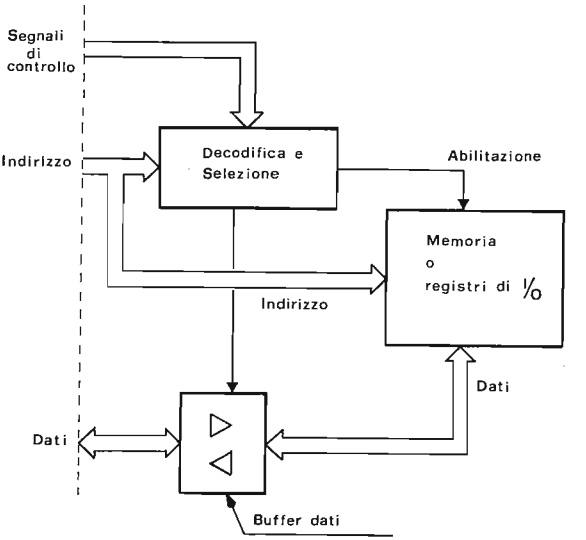


Fig. I.1 - Struttura di un modulo slave

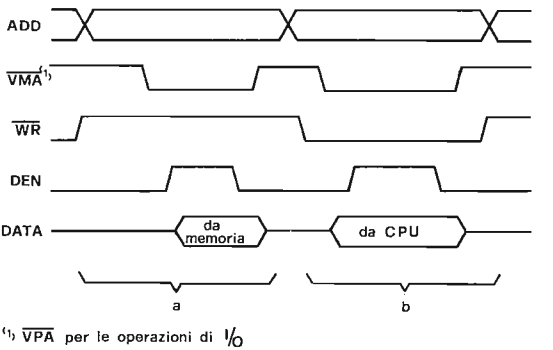


Fig. I.2 - Segnali di controllo dei trasferimenti per i MUBUS

zamento), mentre la combinazione in AND di VMA/VPA e DEN abilita il trasferimento dei dati.

Le operazioni su memoria usano tutte le 16 linee di indirizzo, mentre quelle su periferico ne usano solo 8 (ADD 0-7).

Moduli periferici

Una scheda di I/O può ospitare circuiti LSI speciali per interfacce (ad esempio 8255 Intel, PIO Zilog, i vari UART ecc.), oppure può essere realizzata con dei semplici inte-

grati TTL con funzione di registri e buffer.

In ogni caso su una scheda sono presenti più registri ed occorre quindi decodificare non un singolo indirizzo ma un "blocco".

Per chiarire questo concetto, conviene considerare gli 8 bit di selezione ADD 0-7 divisi in due campi, rispettivamente per individuare la scheda (cioè il blocco di indirizzi), e per selezionare i diversi registri presenti su questa. Nell'esempio di fig. I.3, i 6 bit più significativi (ADD 7-2) individuano la scheda (64 combinazioni possibili), ed i 2 bit ADD0, 1 selezionano i registri interni (4 possibilità). La separazione tra registri di ingresso e di uscita ha luogo tramite la linea WRITE. A questo punto è già delineato lo schema a blocchi di una interfaccia con registri separati (fig. I.4).

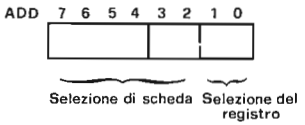


Fig. I.3 - Selezione dei registri di I/O

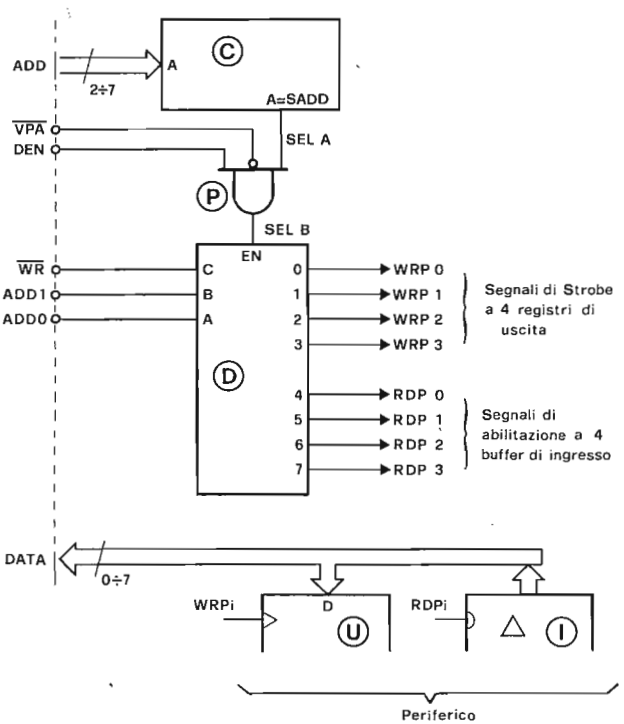


Fig. I.4 - Schema a blocchi di una scheda di interfaccia con un periferico. Di quest'ultimo è indicata solo la parte direttamente con il bus dati.

Il comparatore Ⓒ riconosce, dalle linee ADD 7-2, l'indirizzo di scheda (SADD). La selezione di scheda SELA è validata dai segnali VPA e DEN, e diventa il segnale SELB che abilita il decodificatore Ⓓ il quale a sua volta seleziona i diversi registri: 4 di ingresso (moduli Ⓘ) e 4 di uscita (moduli Ⓤ). Il blocco Ⓒ è essenzialmente un comparatore e può essere realizzato in diversi modi, ad esempio mediante un apposito integrato, con una rete di OR-esclusivi, con uno o più decodificatori, con un AND e degli inverter e così via. Alcune soluzioni circuitali sono indicate in fig. I.5.

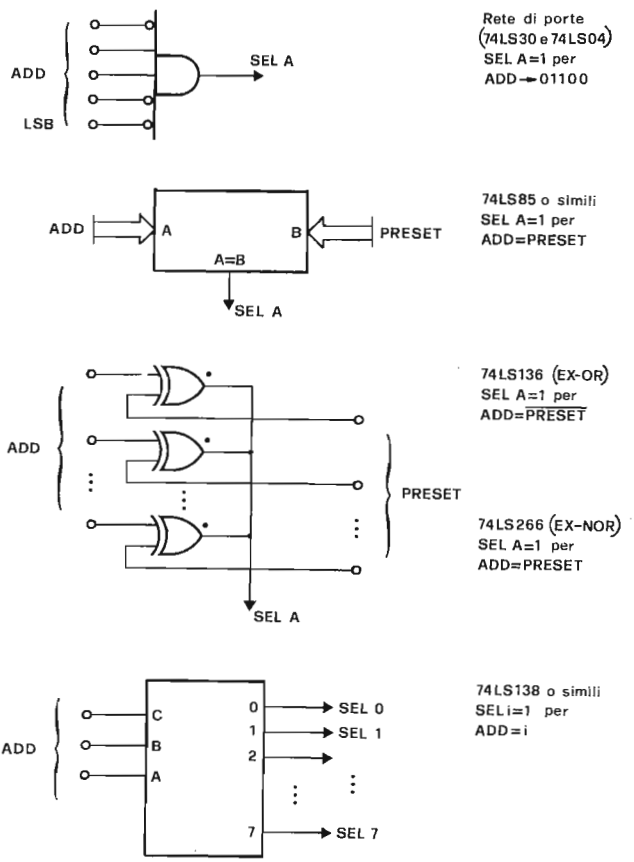


Fig. I.5 - Circuiti per il decodificatore di indirizzo. Nei casi a), b), c) l'indirizzo selezionato è programmabile.

Volendo realizzare delle porte parallelo ciascun registro di ingresso è un buffer 3-stati, abilitato da un segnale RDPi, e ciascun registro di uscita è un latch (gruppo di Flip-Flop D), strobato da un segnale WRPi. Un esempio di semplice interfaccia realizzata con questa tecnica è dato dallo schema di fig. I.6.

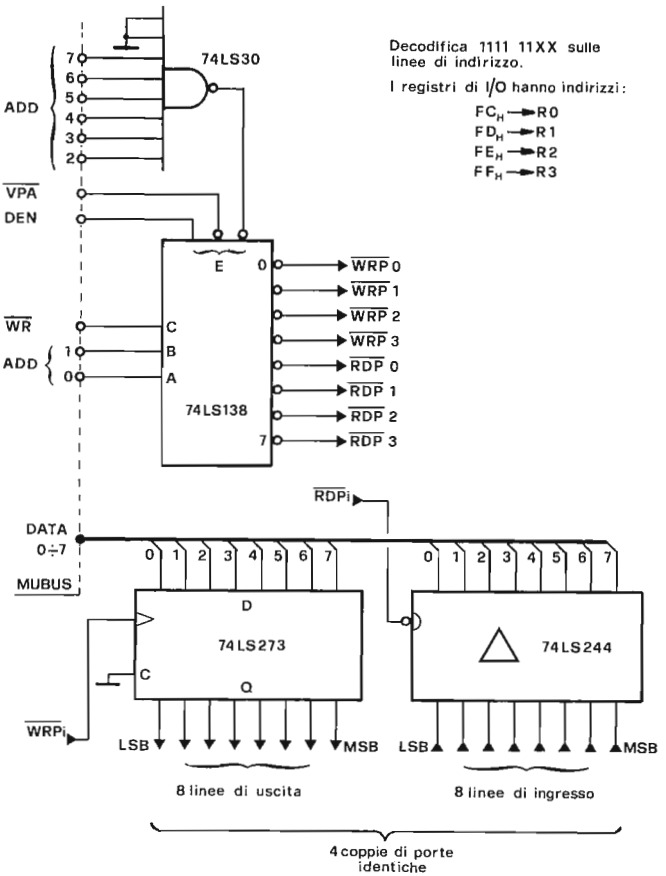


Fig. I.6 - Interfaccia per 4 porte di I/O parallele.

A questa struttura base è possibile aggiungere altri registri per ottenere ingressi con memoria, oppure la logica di controllo per "handshake" con dispositivi esterni, o ancora traslatori di livello o separatori ottici.

Se invece il "cuore" del periferico è un integrato complesso quali i vari PIO, USART e simili citati prima, intervengono due nuovi problemi:

1) Le interfacce complesse comprendono più registri ed hanno in genere circuiti interni di selezione. Questi assorbono in parte le funzioni del decoder Ⓓ e della Logica Ⓟ di fig. I.4.

2) Questi circuiti sono realizzati in tecnologia MOS ed hanno quindi un basso fan-out; possono essere direttamente collegati alle linee dati solo in sistemi molto piccoli, che presentano un carico limitato sul bus. Per prevedere la possibilità di espansione è buona norma interporre sempre dei buffer TTL.

I segnali di abilitazione e selezione delle interfacce LSI sono organizzati secondo due tecniche base, entrambe facilmente interfacciabili ai segnali MUBUS.

Nel primo caso (fig. I.7a), sono presenti un piedino di abilitazione (CE), un comando di direzione (WR/$\overline{RD}$), per specificare se l'operazione è di lettura o di scrittura, ed una o più linee di indirizzo per la selezione dei registri interni. Questa è la tecnica usata ad esempio per le interfacce Zilog (PIO, SIO, CTC) e Motorola (PIA, ACIA,...), che sono in pratica collegabili direttamente al MUBUS.

Nel secondo caso (fig. I.7b), oltre al segnale di abilitazione sono presenti i due segnali RD e WR, analoghi a RDPi e WRPi di fig. I.4, che attivano rispettivamente operazioni di lettura e di scrittura sui registri interni selezionati con le linee di indirizzo.

L'interfaccia verso MUBUS richiede semplicemente una logica di decodifica per ricavare RDP e WRP dalle linee WR e DEN. Questa soluzione è tipica della famiglia Intel (8255, 8251).

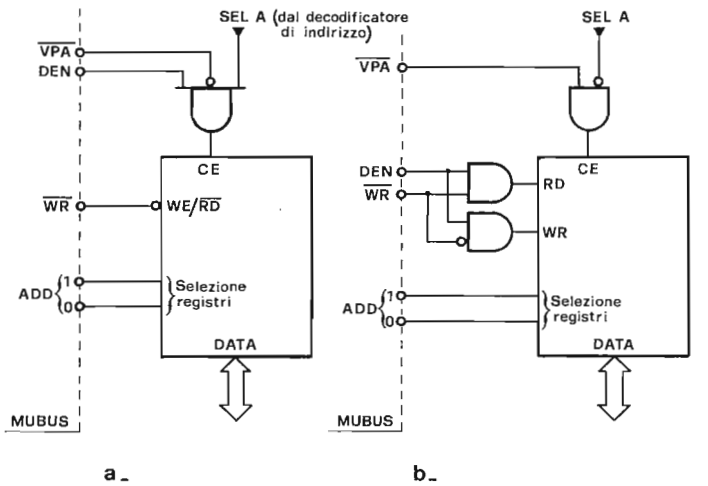


Fig. I.7 - Segnali di comando per controllori di periferici complessi

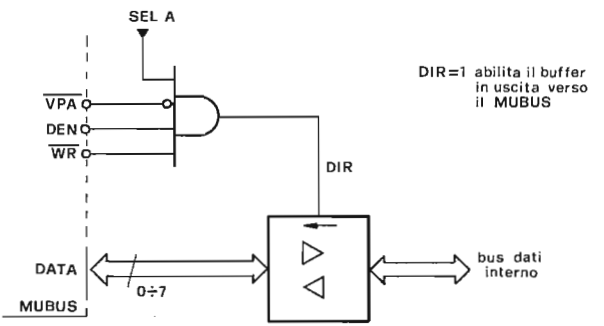


Fig. I.8 - Logica di abilitazione dei buffer dati.

I buffer verso le linee dati devono essere bidirezionali e gestiti in modo da non determinare conflitti di accesso sul bus (abilitazione contemporanea di più buffer 3-stati connessi alle stesse linee). Questa condizione è soddisfatta se l'abilitazione ad uscire verso il bus viene data solo quando la scheda è selezionata ed è in corso una operazione di lettura (WRITE = 1). La logica di controllo è pertanto quella di fig. I.8.

Moduli di memoria

Anche le schede con espansioni di memoria sono del tipo slave e quindi hanno struttura analoga a quella di fig. I.1. La scheda di memoria contiene uno o più "banchi", indicando con questo termine il gruppo di integrati con parallelismo pari alla dimensione di parola del sistema. Ad esempio, per sistemi a 8 bit, un banco può essere formato con 8×2102 (1K × 1 bit) oppure 2×2114 (1K × 4 bit). Tutti gli integrati di uno stesso banco devono avere un'unico comando di abilitazione.

Analogamente a quanto fatto per i periferici (fig. I.3), dividiamo anche i 16 bit di indirizzo della memoria in diversi campi, rispettivamente per selezionare una scheda, un determinato banco tra quelli presenti sulla scheda, e la specifica cella tra tutte quelle del banco. La fig. I.9 riporta un esempio di assegnazione dei diversi campi; in base a questa ciascuna linea di indirizzo deve essere portata al circuito adatto (selezione di scheda, di banco, di cella).

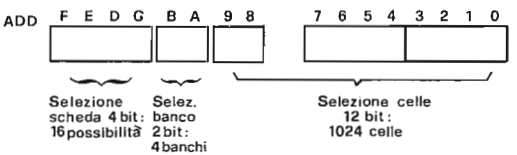


Fig. I.9 - Selezione delle celle di memoria: esempio per una scheda da 4 Kbyte (4 banchi da 1 K ciascuno).

Per quest'ultima si ha una connessione diretta al circuito integrato di memoria ma, dato che nella scheda di memoria si hanno più integrati connessi in parallelo sulle linee dati ed indirizzi, è necessario interporre buffer TTL su tutte le linee per limitare il carico sul bus.

La linea WR non deve selezionare dispositivi diversi, ma va usata come comando di scrittura per memorie RAM, e viene ignorata per memorie ROM (o comunque a sola lettura).

Analogamente a quanto visto nel caso dei periferici, anche nel caso delle memorie è possibile individuare alcune strutture base di selezione e comando, sostanzialmente analoghe a quelle di fig. I.7. Per i circuiti integrati di memoria in realtà le combinazioni diventano quattro, perchè le linee dati possono essere bidirezionali o separate tra ingresso ed uscita. (Vedi in proposito "Sistemi a microprocessore", Boringhieri, cap. 4).

Lo schema base di una scheda di memoria utilizza gli stessi blocchi funzionali già definiti per i periferici (fig. I.10):

- decodificatore dell'indirizzo di scheda; Ⓒ
- logica di validazione; Ⓟ
- logica di selezione dei banchi; Ⓓ

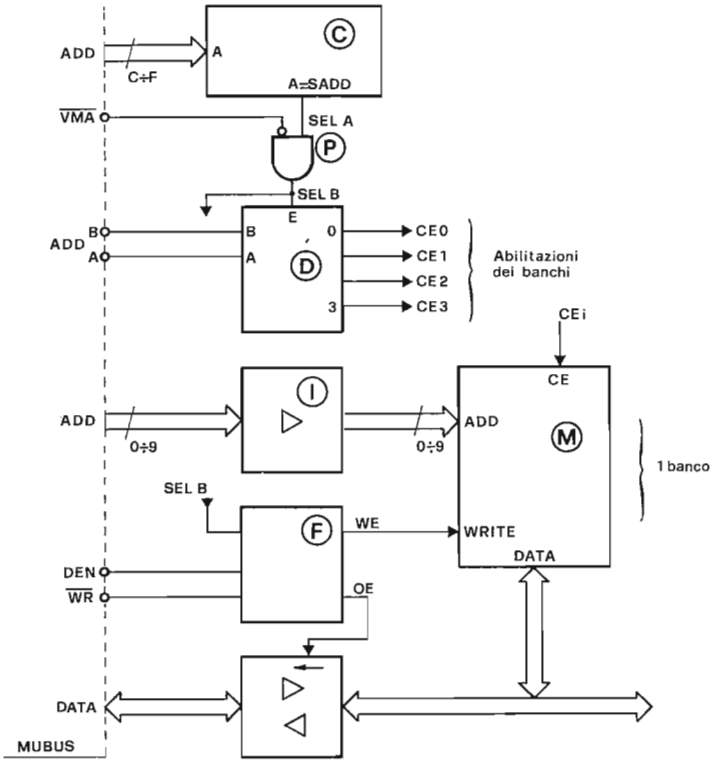


Fig. I.10 - Schema a blocchi di una scheda di memoria; i bit di indirizzo sono usati come indicato in fig. I.9.

- logica di controllo dei buffer dati; Ⓕ
- buffer indirizzi; Ⓘ
- banchi di memoria; Ⓜ
- buffer dati; Ⓑ

Come unica differenza rispetto alla fig. I.4, si noti che ora la selezione è validata dal solo segnale VMA (anzichè dall'AND di VPA e DEN), e che DEN è usato esclusivamente dalla logica di comando della scrittura e per abilitare il buffer dati. Questo permette di anticipare la selezione della cella di memoria e quindi aumenta il tempo di ciclo effettivamente disponibile per il circuito integrato di memoria.

La linea di DEN può essere usata per operazioni particolari che richiedano di disabilitare la memoria di sistema (ad esempio per inserire "memorie fantasma"). Per eseguire quest'ultima operazione basta portare la linea DEN a 0 (i buffer dati della memoria di sistema sono cosí disabilitati ed il bus dati rimane libero), ed inserire una memoria sostitutiva (detta appunto "fantasma"), che contiene il programma di inizializzazione. Al termine di questa operazione si disabilita la memoria fantasma e si rilascia la linea di DEN, riportando il sistema in condizioni normali di funzionamento.

La fig. I.11 riporta, a titolo di esempio, lo schema di una scheda da 4 Kbyte realizzata con gli integrati 2114 (1 K × 4 bit). Lo schema presenta alcune caratteristiche particolari:

- i buffer dati sono invertenti, per usare integrati più veloci e di minor consumo (8226 anzichè 8216); naturalmente questa inversione è del tutto invisibile dal bus (i dati sono invertiti due volte, in scrittura ed in lettura).
- non sono presenti buffer indirizzi; la cosa è accettabile solo per sistemi composti da poche schede.

Problemi di velocità

Fino a questo punto non ci siamo occupati della temporizzazione; in realtà nel progetto delle memorie e dei periferici occorre garantire che siano rispettate in ogni possibile situazione le specifiche dei costruttori sui vari tempi di ciclo, di accesso, di scrittura e così via.

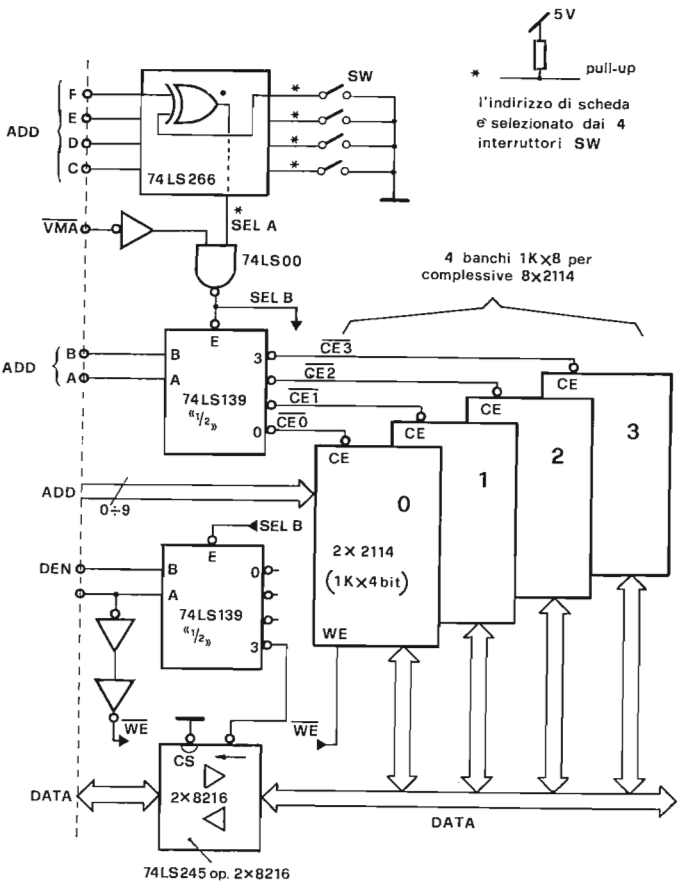


Fig. I.11 - Scheda di memoria da 4Kbite.

Possiamo ad esempio determinare il tempo di accesso effettivamente richiesto dai circuiti di memoria per lo schema di fig. 1.11.

Occorre per questo tenere conto di tutti i ritardi che si incontrano partendo dalla richiesta di trasferimento (originata dalla CPU), fino al momento in cui i dati sono effettivamente disponibili ai piedini dell'unità centrale.

Facendo riferimento alla fig. 1.12 si possono determinare tutti i ritardi via via incontrati su questo percorso:

T 1 - ritardo della logica di codifica tra CPU e bus di sistema;
T 2 - ritardo dei buffero comandi della scheda CPU;
T 3 - ritardo della logica di decodifica dell'indirizzo sulla scheda di memoria;
T 4 - tempo di accesso degli integrati di memoria (cioè ritardi della decodifica e dei buffer interni);
T 5 - ritardo di propagazione dei buffer dati sulla scheda di memoria;
T 6 - ritardo di propagazione dei buffer dati sulla CPU.

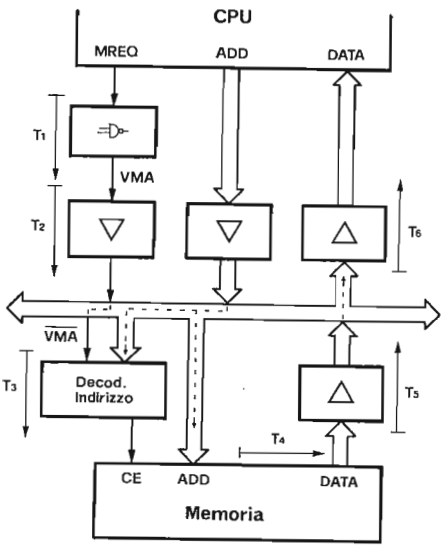


Fig. I.12 - Calcolo del tempo di accesso.

In questo modo, noto il tempo di accesso richiesto dalla CPU, ed i vari ritardi, è possibile determinare il tempo di accesso effettivamente richiesto per i chip di memoria.

Se il tempo di accesso dei chip di memoria è $Tacc_H$, il tempo di accesso effettivo ai piedini della CPU è:

$$Tacc_E = Tacc_H + T_1 + T_2 + T_3 + T_5 + T_6.$$

Questo tempo deve essere inferiore al tempo di accesso specificato dal costruttore per la CPU.

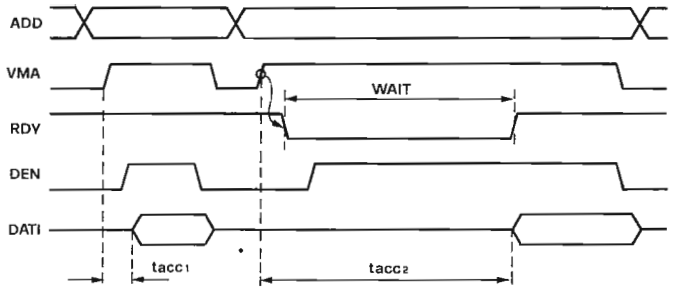


Figura I.13 - Cicli di trasferimento con stati di WAIT

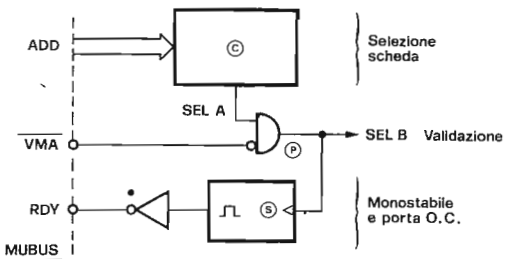


Figura I.14 - Logica di comando della linea RDY per una scheda di memoria

Qualora non fossero disponibili circuiti di velocità adeguata, occorre rallentare le operazioni dell'unità centrale usando il segnale di RDY, che forza la CPU nello stato di WAIT, (attesa), prolungando così la durata dei cicli di trasferimento (Fig. I.13). Questo segnale sarà quindi attivato da quei moduli slave la cui tempistica non è adeguata alla velocità dell'unità centrale. La logica di controllo comprende di solito un monostabile S in fig. I.14 comandato dal segnale di selezione della piastra. La linea RDY può essere attivata da più moduli, e pertanto deve essere pilotata con porte a collettore aperto. La stessa tecnica può servire per sincronizzare le operazioni dell'unità centrale con eventi esterni (ad esempio la disponibilità di un dato su una porta di ingresso), forzando la CPU nello stato di WAIT fino al momento in cui è possibile concludere l'operazione.

Nella fig. I.15 questa tecnica è usata, a titolo di esempio, per gestire un convertitore Analogico/Digitale; l'operazione di lettura fa partire un ciclo di conversione (segnale CONVST), ed il nuovo dato è disponibile solo dopo che questa è stata conclusa (segnale EOC = End Of Conversion). Il processore rimane quindi bloccato nello stato di

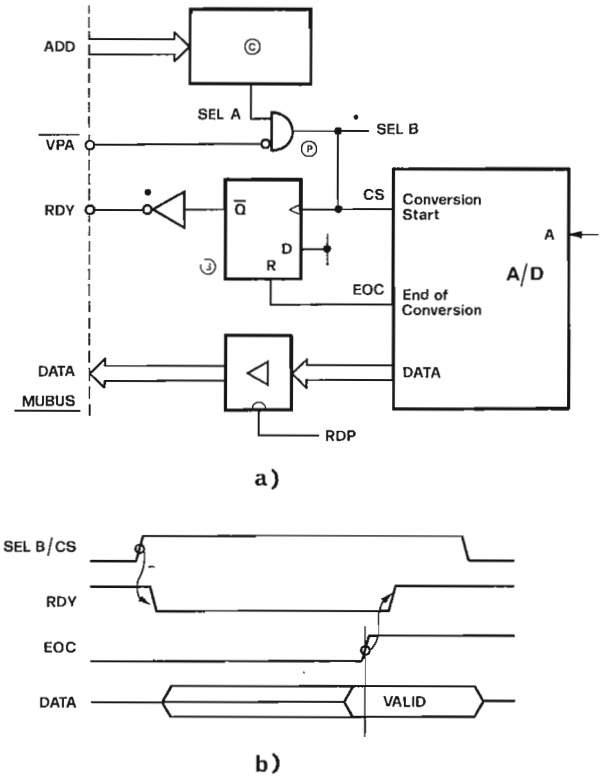


Fig. I.15 - Logica di comando e temporizzazione della linea RDY per la sincronizzazione con un periferico (convertitore A/D).

WAIT fino a quando non è possibile eseguire correttamente l'operazione di lettura del nuovo dato.

Se il sistema contiene memorie dinamiche, rinfrescate dalla CPU, la durata dello stato di WAIT deve essere limitata, perchè in condizione di WAIT lo Z 80 non genera i cicli di rinfresco.

ISTRUZIONI Z80© 1978 DDC

LD	B	C	D	E	H	L	(HL)	A	
B	40	41	42	43	44	45	46	47	06,n
C	48	49	4A	4B	4C	4D	4E	4F	0E,n
D	50	51	52	53	54	55	56	57	16,n
E	58	59	5A	5B	5C	5D	5E	5F	1E,n
H	60	61	62	63	64	65	66	67	26,n
L	68	69	6A	6B	6C	6D	6E	6F	2E,n
(HL)	70	71	72	73	74	75	-	77	36,n
A	78	79	7A	7B	7C	7D	7E	7F	3E,n
ADD	80	81	82	83	84	85	86	87	C6,n
ADC	88	89	8A	8B	8C	8D	8E	8F	CE,n
SUB	90	91	92	93	94	95	96	97	D6,n
SBC	98	99	9A	9B	9C	9D	9E	9F	DE,n
AND	A0	A1	A2	A3	A4	A5	A6	A7	E6,n
XOR	A8	A9	AA	AB	AC	AD	AE	AF	EE,n
OR	B0	B1	B2	B3	B4	B5	B6	B7	F6,n
CP	B8	B9	BA	BB	BC	BD	BE	BF	FE,n
INC	04	0C	14	1C	24	2C	34	3C	
DEC	05	0D	15	1D	25	2D	35	3D	
IN	*40	*48	*50	*58	*60	*68	-	*78	DB,n
OUT	*41	*49	*51	*59	*61	*69	-	*79	D3,n

LD	→ A	← A
(BC)	0A	02
(DE)	1A	12
(HL)	7E	77
(nn)	3A,nn	32,nn
I	*57	*47
R	*5F	*4F

LD	← nn	← (nn)	→ (nn)	HL	PUSH	POP
B C	01,nn	ED,4B,nn	ED,43,nn		C5	C1
D E	11,nn	ED,5B,nn	ED,53,nn		D5	D1
H L	21,nn	24,nn	22,nn		E5	E1
S P	31,nn	ED,7B,nn	ED,73,nn	F9		
A F					F5	F1

	BC	DE	HL	SP
ADD	09	19	29	39
ADC	*4A	*5A	*6A	*7A
SBC	*42	*52	*62	*72
INC	03	13	23	33
DEC	0B	1B	2B	3B

	I	IR	D	DR
LD	*A0	*B0	*A8	*B8
CP	*A1	*B1	*A9	*B9
IN	*A2	*B2	*AA	*BA
OUT	*A3	*B3	*AB	*BB

(DE) ← (HL)
A ↔ (HL)
(HL) ←
← (HL)

EX	AF	rp	HL	
	08	D9	EB	DE
			E3	(SP)

IX ← DD.
IY ← FD.
* → ED

	A		cy	Reg.
RLC	07	00		
RRC	0F	08		
RL	17	10		CB, Reg
RR	1F	18		
SCA		20		
SRA		28		
SLI		30		
SRL		38		
BIT		40		
RES		80		CB, , Reg, bit
SET		C0		

cond.	JP add.	JR disp.	CALL add.	RET
—	C3,nn	18,e	CD,nn	C9
NZ	C2,nn	20,e	C4,nn	C0
Z	CA,nn	28,e	CC,nn	C8
NC	D2,nn	30,e	D4,nn	D0
C	DA,nn	38,e	DC,nn	D8
P.O.	E2,nn		E4,nn	E0
P.E.	EA,nn		EC,nn	E8
P	F2,nn		F4,nn	F0
M	FA,nn		FC,nn	F8

RST	
0	C7
1	CF
2	D7
3	DF
4	E7
5	EF
6	F7
7	FF

DJNZ	10,e
JP (HL)	E9

IM 0	*46
IM 1	*56
IM 2	*5E

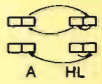
RETI	*4D
RETN	*45

DAA	27
CPL	2F
CCF	3F
SCF	37
NOP	00
HLT	76
DI	F3
EI	FB

Reg.	B	C	D	E	H	L	(HL)	A	
	0	1	2	3	4	5	6	7	bit
	00	08	10	18	20	28	30	38	

RLD	*6F
RRD	*67

NEG	*44



L. 9.000 Cod. 330D ISBN 88-7056-106-2