LA PROGRAMMAZIONE IN ASSEMBLER

Iniziamo da questo numero la pubblicazione mensile di un corso completo di programmazione relativa ai chip della famiglia PIC. Chi meglio di noi, primi in Italia a presentare questi microprocessori, poteva proporre un servizio del genere?

Andrea Sbrana - 1ª parte
su gentile concessione della Microlabs

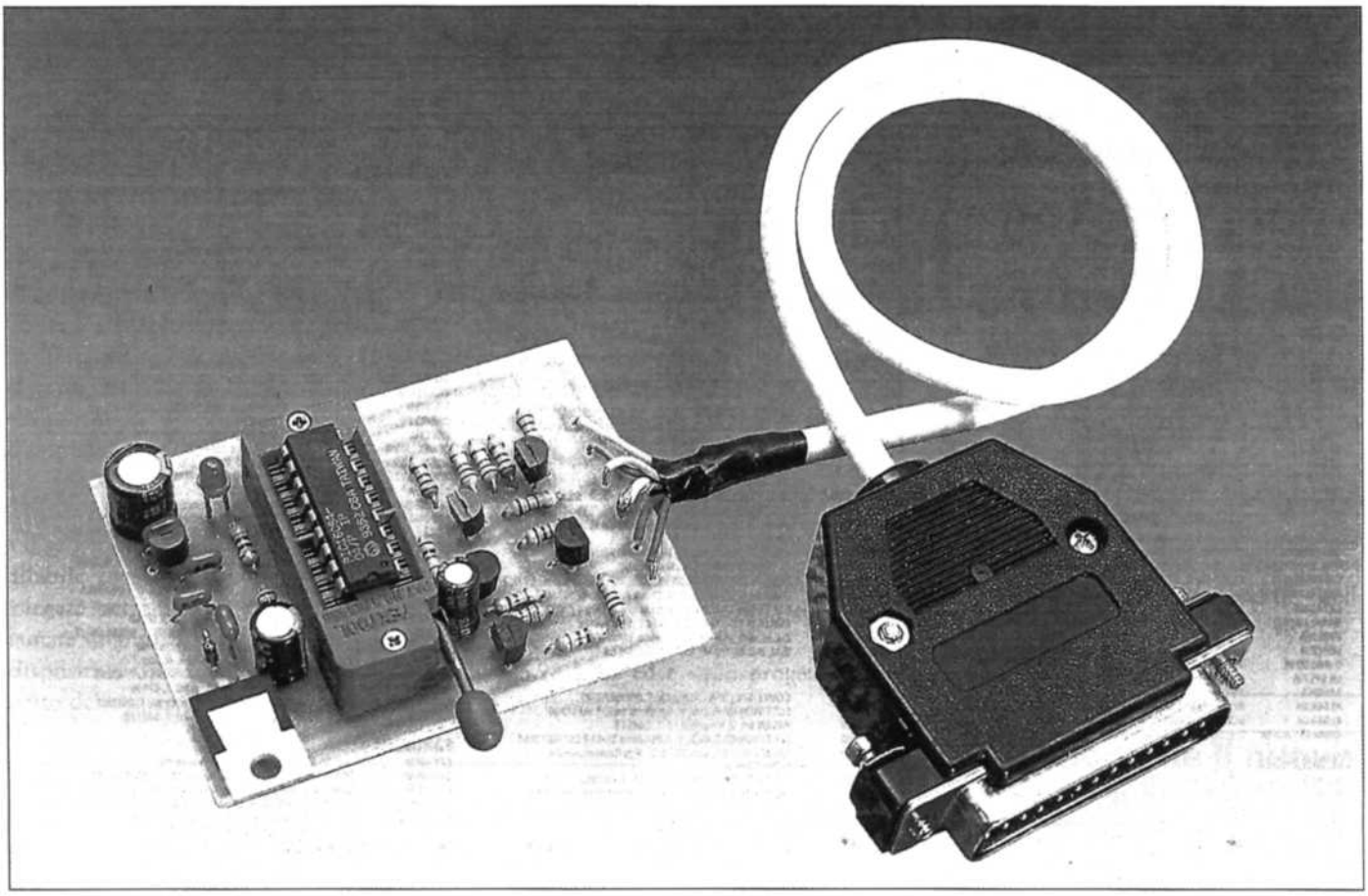
L'evoluzione della tecnologia ha fatto sì che i microcontroller, fino a pochi anni fa completamente sconosciuti, siano oggi abitualmente impiegati non solo in apparati commerciali, ma anche in circuiti dedicati e realizzati da hobbisti e sperimentatori, data la facilità con cui vengono programmati.

Ma la maggior parte degli hobbisti elettronici, tradizionalmente ha sempre avuto una certa difficoltà con il software, indispensabile per poter sfruttare questi chip, quindi abbiamo deciso di dedicare una serie di puntate alla programmazione in assembler che, anche se finalizzata al PIC16C84 della Microchip, risulterà molto utile per l'apprendimento di concetti che sono al di sopra del componente impiegato.

In questa prima puntata, illustreremo un programmatore per PIC16C84 molto semplice, da collegare alla porta parallela di un qualsiasi computer IBM compatibile e di costo estremamente ridotto, dato l'esiguo numero di componenti.

Questo prodotto ci è stato gentilmente concesso dalla ditta Microlabs, Via Circonvallazione, 11 20060 Masate (MI), telefono 02/95762168 oppure 0336/408090 che si è resa pienamente disponibile anche per risolvere eventuali problemi dei nostri lettori relativi a qualsiasi tipo di microcontroller della Microchip.

A tale proposito potrete telefonare direttamente all'Ing. Gordan Rancic, che segue costantemente l'evoluzione della famiglia PIC.



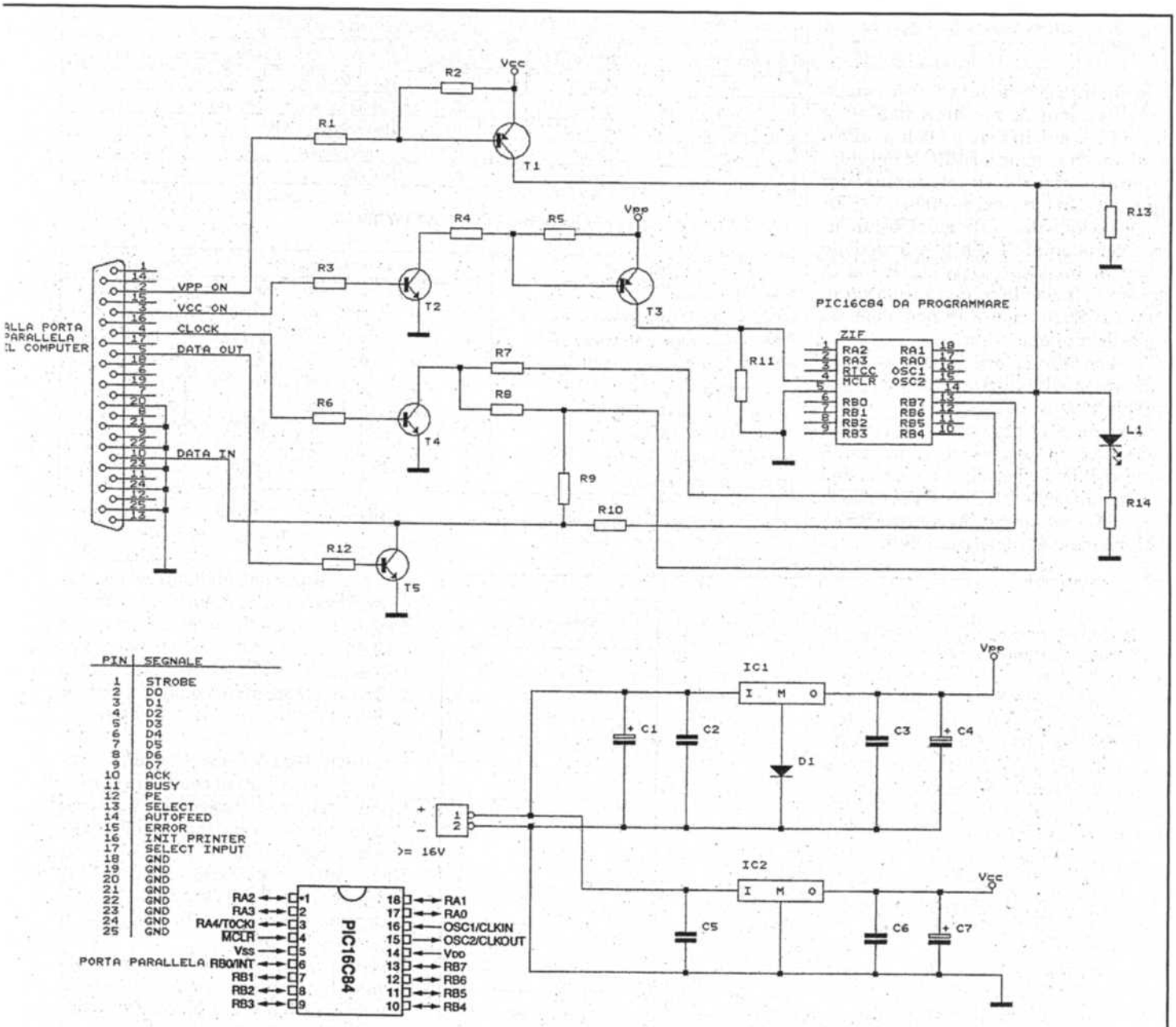


Figura 1. Schema elettrico del programmatore

Funziona così

Passiamo quindi a vedere le caratteristiche del programmatore, iniziando dallo schema elettrico di Figura 1.

Dalla porta parallela del computer, vengono prelevati 4 segnali (D0, D1, D2 e D3) che comandano le funzioni del programmatore, mentre un segnale entra nel computer (ACK) per avere la possibilità di uno scambio dati seriale.

Il segnale D0 viene impiegato per abilitare la tensione di alimentazione Vcc per il 16C84 attraverso il transistor T1.

Quando D0 è a basso livello, sul pin 14 del PIC è presente una tensione di 5 volt.

La tensione di programmazione di 12,8 V invece, viene abilitata con il segnale D1 per mezzo dei due transistor T2 e T3.

I due segnali D3 e D4 simulano rispettivamente il segnale di clock e di dato per il PIC.

I dati che provengono dal PIC, sono invece inviati sulla linea ACK del computer per mezzo di three-state implementato con il transistor T5.

Le due alimentazioni per la programmazione vengono stabilizzate da due regolatori appositi, IC1 e IC2. Si nota che IC1 ha il riferimento di massa elevato con un diodo, per ottenere i 12,8 V necessari alla tensione di programmazione.

Per questo motivo, la tensione di ingresso del programmatore non potrà mai stare al di sotto di 16 V, pena il non corretto funzionamento del circuito.

Sempre in Figura 1 vediamo le connessioni relative al connettore DB25 della porta parallela e quelle relative alla piedinatura del PIC16C84.

La programmazione del PIC

In Figura 2 possiamo vedere la mappa delle locazioni di memoria impiegate nel 16C84: dall'indirizzo 000h al 3FFh troviamo la memoria EPROM del chip, ovvero dove risiederà il programma vero e proprio. Gli indirizzi da 400h a 1FFFh non sono implementati, mentre dall'indirizzo 2000h al 200Fh troviamo delle locazioni in parte scrivibili, in parte riservate, fra cui la configuration word.

Poi, da 2010h a 3FFFh non abbiamo più celle implementate.

Il programmatore dovrà, quindi, lavorare sugli indirizzi 000h-3FFh e 1FFFh-2000h.

Durante la programmazione, i pin del PIC16C84 interessati sono soltanto 5, come visibile in Tabella 1a, e cioè i due di alimentazione, il MCLR, l'RB6 e l'RB7.

L'RB6 accetta il clock, mentre l'RB7 è bidirezionale e trasferisce i dati.

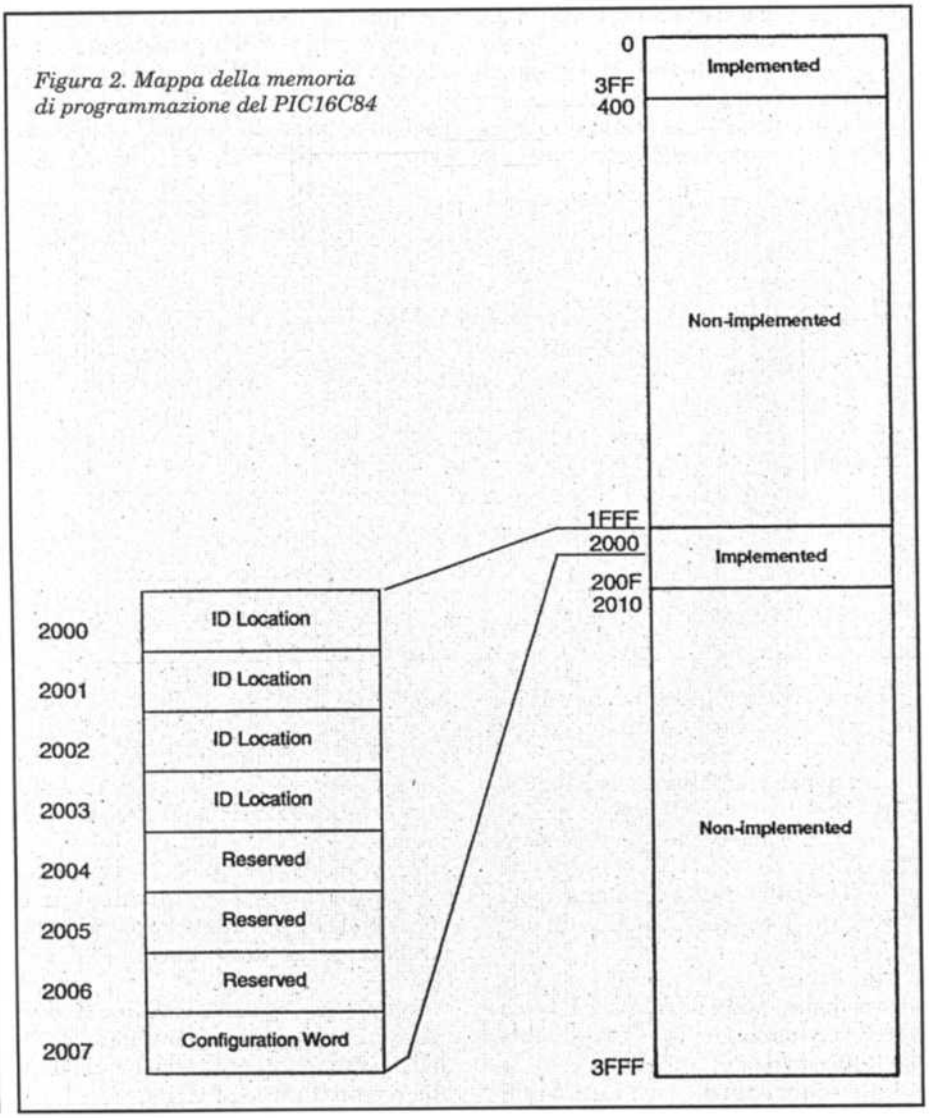
Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RB6	CLOCK	I	Clock input
RB7	DATA	I/O	Data input/output
$\overline{MCLR}$	VTEST MODE	P*	Program Mode Select
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Tabella 1a. Segnali per la programmazione del PIC16C84

Command	Mapping (msb ... lsb)						Data
Load Configuration	0	0	0	0	0	0	0, data (14), 0
Load Data for Program Memory	0	0	0	0	1	0	0, data (14), 0
Read Data from Program Memory	0	0	0	1	0	0	0, data (14), 0
Increment Address	0	0	0	1	1	0	
Begin Programming	0	0	1	0	0	0	
Load Data for Data Memory	0	0	0	0	1	1	0, data (14), 0
Read Data from Data Memory	0	0	0	1	0	1	0, data (14), 0
Bulk Erase Program Memory	0	0	1	0	0	1	
Bulk Erase Data Memory	0	0	1	0	1	1	

Tabella 1b. Mappa dei comandi implementati nel PIC16C84

Figura 2. Mappa della memoria di programmazione del PIC16C84



Il MCLR permette di entrare in programmazione semplicemente in funzione della tensione che vede in ingresso (5 o 12,8).

I comandi che è possibile inviare a PIC sotto programmazione sono visibili in Tabella 1b.

Load Configuration: permette di caricare il registro di configurazione
Load Data for Program Memory: consente di caricare una cella di memoria del programma
Read Data from Program Memory: legge una cella di memoria
Increment Address: fa avanzare il program counter
Begin Programming: fa iniziare la fase di programmazione di una cella
Load Data for Data Memory: carica una cella della memoria dati
Read Data from Data Memory: legge una cella della memoria dati
Bulk Erase Program Memory: cancella la memoria di programma
Bulk Erase Data Memory: cancella la memoria dati.

Il protocollo di invio e ricezione dati, è visibile rispettivamente in Figura 3a e 3b.

Come si vede chiaramente, tutte le informazioni passano sui due segnali RB6 (clock) e RB7 (data).

In Figura 5 vediamo che cosa dovremo scrivere nella configuration word per settare il modo di funzionamento del PIC: i bit 0 e 1 indicano il tipo di oscillatore scelto, il bit 2 se il watchdog deve

essere abilitato o meno, il bit 3 se vogliamo inserire un ritardo di 65 mS ad ogni partenza del chip (per un reset più sicuro) ed, infine, il bit 4 deve essere impostato per abilitare o meno la protezione contro letture non permesse del programma.

Uno sguardo al software di programmazione

Il programma che va inserito nel computer per poter sfruttare la nostra interfaccia, è free-ware, ovvero può essere copiato da tutti, ma non per essere rivenduto.

A tale proposito, per ottenerlo potrete telefonare alla Microlabs, che potrà inviarvelo gratuitamente (escluso chiaramente il costo del dischetto e delle spese postali) corredato eventualmente del circuito stampato e di alcuni PIC per le vostre prove che farete durante il corso di programmazione.

Ricordiamo che il PIC16C84 è elettricamente cancellabile e riscrivibile, quindi anche se sbaglierete un programma, potrete riprogrammare il chip senza necessità di lampade per EEPROM.

Schermata 1

```
:    PRO84 - Microchip PIC 16C84 programmer Driver                    :
: Copyright (c) MicroLabs 1995. All rights reserved.                  :
:    Written by Gordan RANCIC for PROGETTO ELEKTOR                    :
:    Shareware version 1.0 - 10. September 1995                       :

USAGE: PRO84 <FILENAME> /arguments... /....
<FILENAME> is the name of the DOS file for READ/WRITE in INHX8M format
ARGUMENTS:  /P       = program                          default
            /F       = program fuses only
            /C       = erase device only
            /E       = eeprom program only
            /NOEE    = do not program/read eeprom
            /NOID    = Do not program/verify ID code
            /R       = read device to file
            /NB      = no blank check                   BLANK CHECK
            /B       = blank check only
            /V       = verify only
            /OSC:XT  = oscillator type LP/RC/XT/HS      XT
            /WD:ON   = watch dog ON/OFF                 ON
            /CP:ON   = code protection                  OFF
            /TU:ON   = power up timer OFF/ON            ON
            /LPT2    = paralel port LPT1, LPT2          LPT1
```

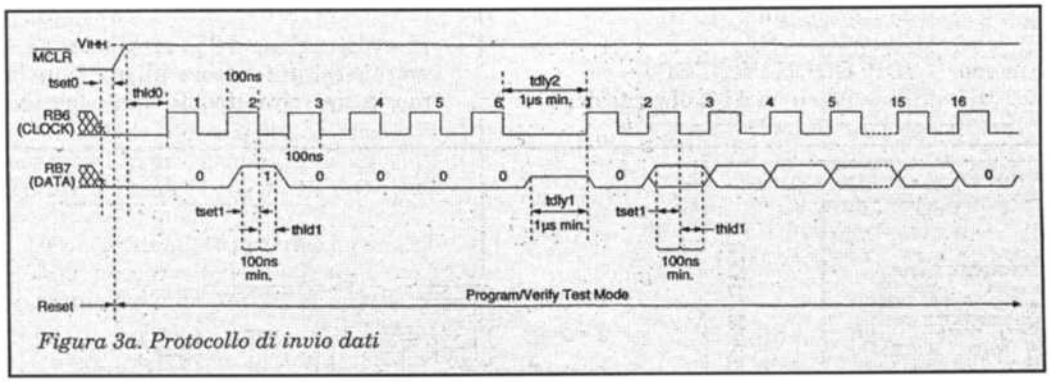


Figura 3a. Protocollo di invio dati

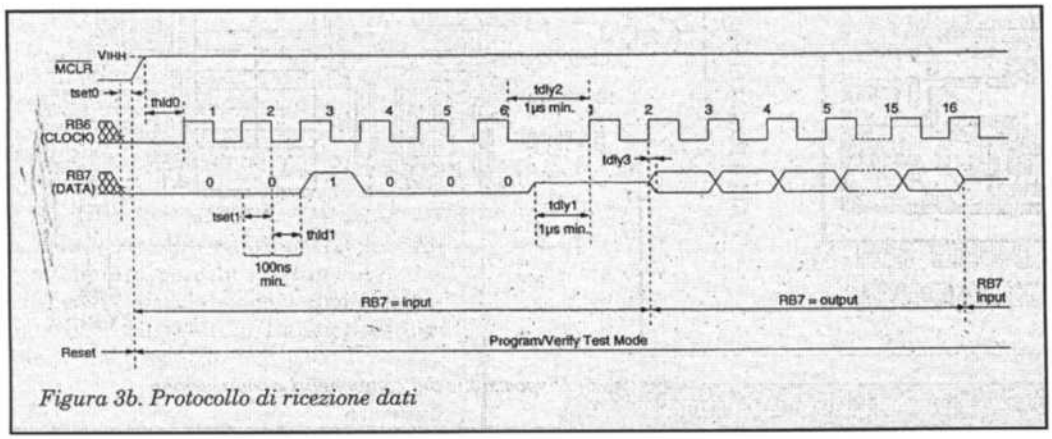


Figura 3b. Protocollo di ricezione dati

Prima di vedere la sintassi di programmazione ricordiamo due cose: il software di programmazione è stato realizzato per tutti i tipi di computer, quindi anche per i più lenti. Se impiegherete 486DX ad elevata velocità oppure pentium, dovrete togliere l'opzione "Turbo".

Inoltre il software per l'assemblaggio del codice sorgente, è anch'esso sharewware e disponibile assieme a questo programma.

La ditta Microlabs inoltre, ha implementato un simulatore grafico che potrete richiedere ad un prezzo molto basso, specificando di essere lettori di Progetto.

Se digitate al prompt il solo comando:

> PRO84

apparirà la **Schermata 1**, visibile in questa stessa pagina.

Tutti i parametri di programmazione possono essere inseriti senza rispettare un ordine particolare se non quello relativo al nome del file, che deve essere inserito per primo.

Il file, per essere accettato dal programmatore, dovrà essere in formato INTEL HEX 8 bit (INHX8M).

Vediamo allora i parametri:

/P PROGRAM
Viene implementato di default e comanda la programmazione del chip

/F PROGRAM FUSES
Questo parametro comanda la sola programmazione dei fusibili, ovvero della configuration word

/C ERASE DEVICE
Permette di cancellare l'intera memoria del chip

Bit Number:	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	CP	PWRTE	WDTE	FOSC1	FOSC0

bit 4: **CP**, Code Protection Configuration Bit
1 = code protection off
0 = code protection on

bit 3: **PWRTE**, Power Up Timer Enable Configuration Bit
1 = Power up timer enabled
0 = Power up timer disabled

bit 3-2: **WDTE**, WDT Enable Configuration Bits
1 = WDT enabled
0 = WDT disabled

bit 1-0 **FOSC<1:0>**, Oscillator Selection Configuration Bits
11: RC oscillator
10: HS oscillator
01: XT oscillator
00: LP oscillator

Figura 4. Registro dei fusibili di settaggio

/E EEPROM ONLY
Questo parametro consente di abilitare le operazioni sulla sola memoria EEPROM

/NOEE NO EEPROM
Disabilita tutte le operazioni sulla EEPROM

/NOID NO ID
Code Disabilita le operazioni sul codice di identificazione

/R READ
Legge il contenuto di un PIC e lo memorizza in un file

/NB NO BLANK
Elimina il controllo del blank sul chip

/B BLANK ONLY
Esegue solo il controllo del blank

/V VERIFY ONLY
Questo parametro verifica il contenuto del chip con quello del file

/OSC: OSCILLATOR
Consente il settaggio del tipo di oscillatore desiderato

/CP: CODE PROTECT
Inserisce o no il bit di protezione

/WD: WATCH DOG
Abilita o meno il watchdog

/TU: POWER UP TIMER
Abilita o meno il Power-up reset

/LPTx PC PORT
Usato per cambiare il numero della porta parallela

Montaggio

La costruzione del programmatore è semplicissima, specie sfruttando la traccia del circuito stampato che vi

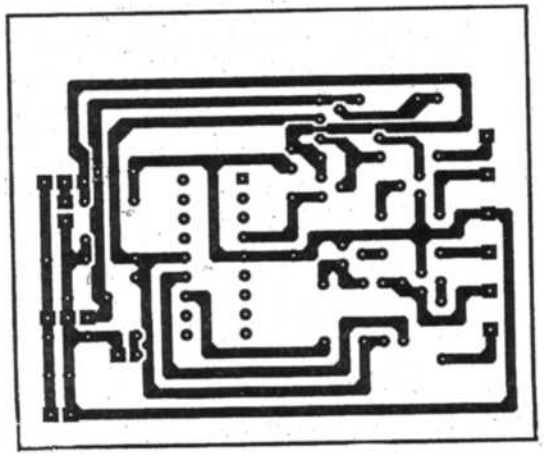


Figura 5. Circuito stampato scala 1:1

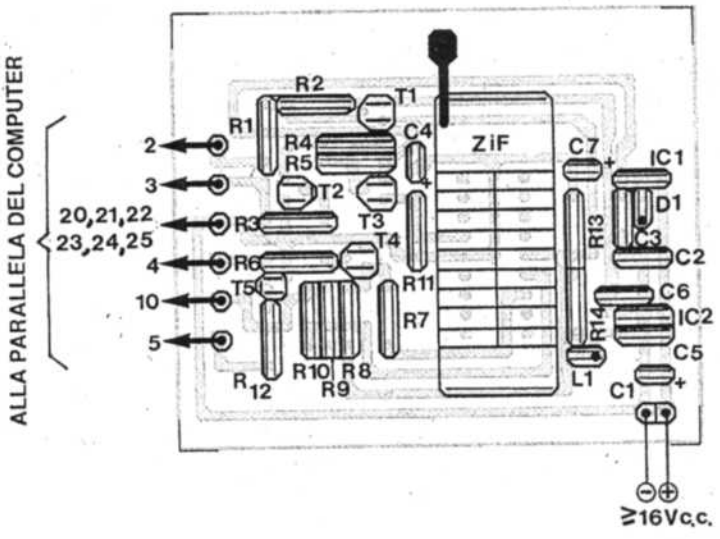


Figura 6. Disposizione dei componenti e connessioni

ELENCO COMPONENTI

Semiconduttori
IC1: 7812
IC2: 78L05
T1, T3: BC307
T2÷T5: BC337
D1: 1N4148
L1: Led

Resistori
R1, R3: 1 kΩ
R2, R5: 4,7 kΩ
R4, R6, R12: 1,5 kΩ
R7, R10: 100 Ω
R8, R9, R11, R13: 10 kΩ
R14: 470 Ω

Condensatori
C1: 100 µF
C2, C3, C5, C6: 100 nF
C4, C7: 47 µF

viene fornita in Figura 5. In Figura 6, invece, troviamo la disposizione dei componenti.

Fate attenzione a non invertire i pin dei componenti polarizzati ed assicuratevi di aver eseguito i collegamenti con la porta parallela del computer come specificato.

Per l'alimentazione dovete procurarvi un alimentatore da almeno 16 volt di uscita e almeno 300mA in continua.

Noi abbiamo impiegato come zoccolo per il PIC in programmazione un TEXTOOL, ma andrà bene anche uno zoccolo convenzionale.

Se però intendete programmare molti chip, necessariamente dovrete montare uno zoccolo a forza di inserzione zero.

Una volta montati tutti i componenti, collegatevi ad un computer e provate a leggere un PIC.

Se il collegamento tra computer e programmatore è perfetto, non ci saranno problemi, viceversa sul monitor del computer apparirà una scritta che dirà che il programmatore non è perfettamente collegato.

Per qualsiasi problema che incontrate nel croso della realizzazione del programmatore potrete liberamente contattare la Microlabs, parlando direttamente con l'Ing Gordan Rancic, il realizzatore del progetto.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Nella prima puntata, abbiamo familiarizzato con un nuovo programmatore di PIC, espressamente sviluppato per far consentire ai nostri lettori di effettuare praticamente gli esercizi proposti; ora incominciamo con la teoria e la pratica dell'assembler

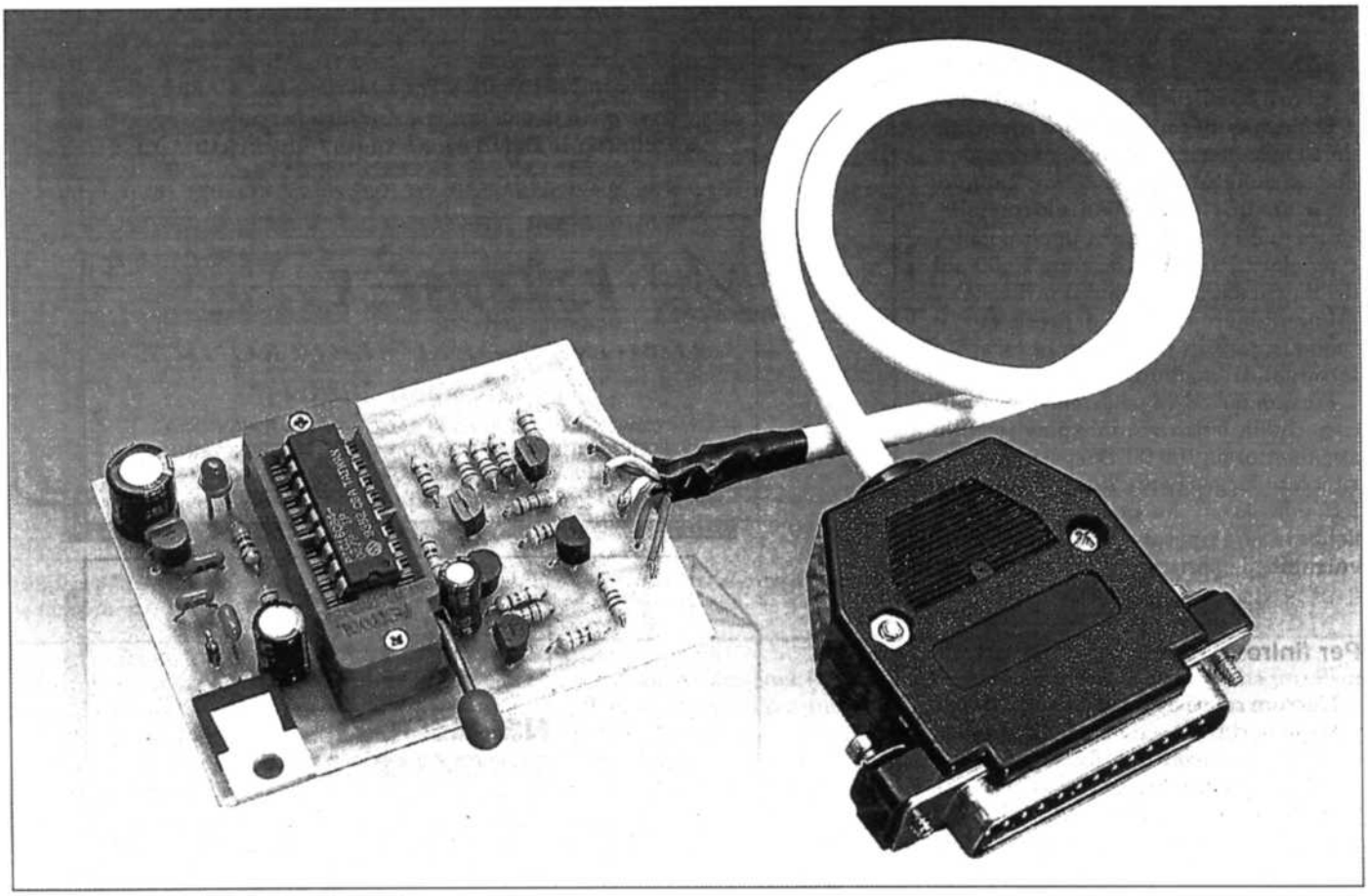
Andrea Sbrana - 2ª parte
su gentile concessione della Microlabs

Dopo aver preso in esame il programmatore low-cost per i soli chip PIC16C84, cercheremo di comprendere come sia fatto internamente uno di questi controller, per capire il perché di una certa istruzione anziché di un'altra.

Sebbene i microcontroller della Microchip partano tutti da una stessa idea progettuale, troviamo alcune differenze tra i vari chip della stessa famiglia e, quindi dovremo soffermarci attentamente su quali siano le peculiarità del PIC16C84

Caratteristiche principali del PIC16C84

- Architettura RISC con 35 istruzioni
- Quasi tutte le istruzioni vengono eseguite in un solo ciclo
- Velocità di 400 nS per istruzione
- 1K EEPROM interna per il programma
- 36 x 8 registri interni general pourpose (SRAM)
- 15 registri special function
- 64 registri ad 8 bit EEPROM per i dati
- 8 livelli di stack
- Indirizzamento diretto, indiretto, relativo
- Quattro sorgenti di interrupt:
  - Pin INT esterno
  - Overflow del TIMER0
  - Cambio di stato sulla porta B <4..7>
  - Fine ciclo di scrittura in EEPROM dati



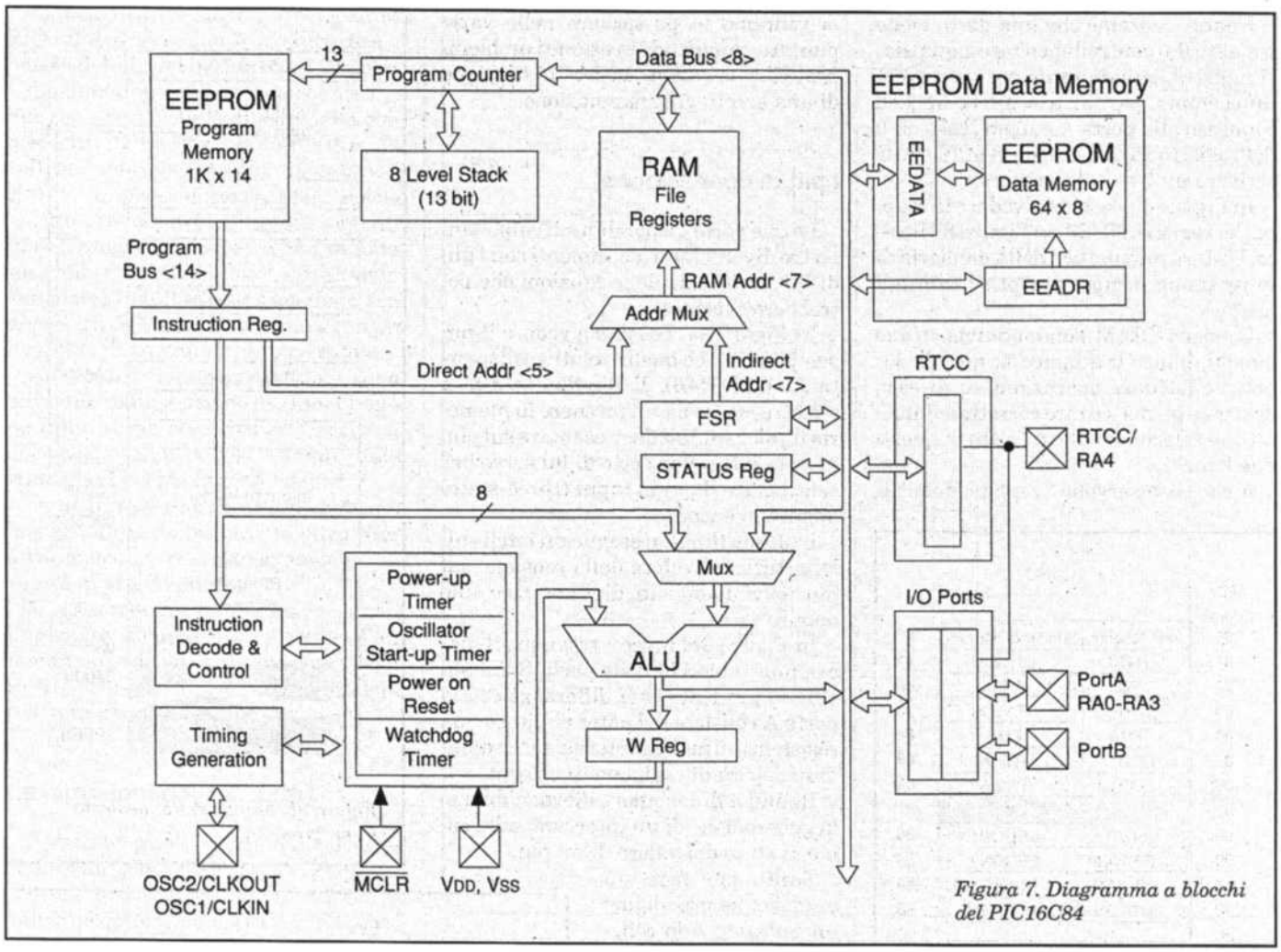


Figura 7. Diagramma a blocchi del PIC16C84

- 1.000.000 di cancellazioni/scritture nella EEPROM memoria dati
- Mantenimento in assenza di tensione > 40 anni
- 13 I/O pin con controllo individuale di direzione
- 20 mA di corrente erogabile per pin
- 25 mA di corrente come sink
- Contatore interno 8 bit + prescaler
- Power-on reset
- Power-up timer
- Watchdog timer
- Protezione del programma in lettura
- Funzionamento in SLEEP
- Oscillatore selezionabile tra 4
- Programmazione seriale in-circuit
- Tensione di alimentazione da 2 a 6 volt
- Assorbimento a 5V e 4MHz < 2mA
- Assorbimento in SLEEP < 1 uA

L'hardware

In Figura 7 vediamo il diagramma a blocchi del PIC16C84. La prima cosa che notiamo è la separazione del bus dati dal bus indirizzi, il che velocizza notevolmente le operazioni.

In alto a sinistra vediamo la Program Memory, ovvero la memoria dove verrà immagazzinato il programma vero e proprio.

L'operazione da eseguire viene indirizzata dal Program Counter, che ha a disposizione uno stack di 8 livelli.

Per coloro che sono a digiuno di controller, diciamo che, durante lo svolgimento di un programma, è possibile richiamare alcune subroutine di uso generale.

Con uno stack ad otto livelli, è possibile richiamare fino ad otto subroutine annidate (ovvero consecutive).

Sul bus dei dati troviamo la Static RAM detta anche File Register.

Questa memoria è di tipo RAM ed è possibile leggere o scrivere qualunque dei suoi registri, che possono essere di uso generale oppure dedicati a particolari funzioni che vedremo nel seguito.

Sempre sul bus dati vediamo connesso il FSR (File Select Register), che potrà servire per indirizzare una cella della SRAM con modalità diversa da quella diretta.

Altro registro impiegato spesso che troviamo sul bus dati è lo STATUS Register, ovvero il registro di stato. Tramite questo registro è possibile sapere l'esito di una istruzione immediatamente dopo che è stata eseguita.

Ma il registro più sfruttato di tutti è senza dubbio il Working Register, detto anche "W" oppure registro di lavoro.

Noterete che nella programmazione, su dieci linee di programma, questo registro viene chiamato almeno tre o quattro volte poiché è l'unico registro che consente il passaggio dei dati da un registro all'altro.

Sempre sul bus dati vediamo anche la ALU, ovvero l'Unità Logico Aritmetica che si occupa di gestire tutte le istruzioni matematiche.

Come periferiche, abbiamo i 64 byte di EEPROM, gestiti dai due registri EEDATA e EEADR, il timer RTCC (Real Time Clock Counter) e i registri delle porte vere e proprie.

Infatti, vedremo che una porta viene trattata dal controller come un qualsiasi registro, quindi, tanto per anticipare un esempio, per far accendere un Led connesso alla porta A sul pin RA0 (bit 0 del registro PORTA A) sarà sufficiente scrivere un 1 su tale registro.

In Figura 8a possiamo vedere la mappa dei registri SRAM, in Figura 8b invece, l'intera mappatura della memoria di programma, compresi gli otto livelli dello stack.

I registri SRAM, sono suddivisi in due banchi, banco 0 e banco 1, quindi per poter effettuare operazioni su di essi, dovremo prima settare correttamente il bit che seleziona l'uno o l'altro di questi due banchi.

A che cosa servono i registri dedicati lo vedremo in particolare nelle varie puntate, prendendo in esame i problemi pratici che possono sorgere in funzione di una errata programmazione.

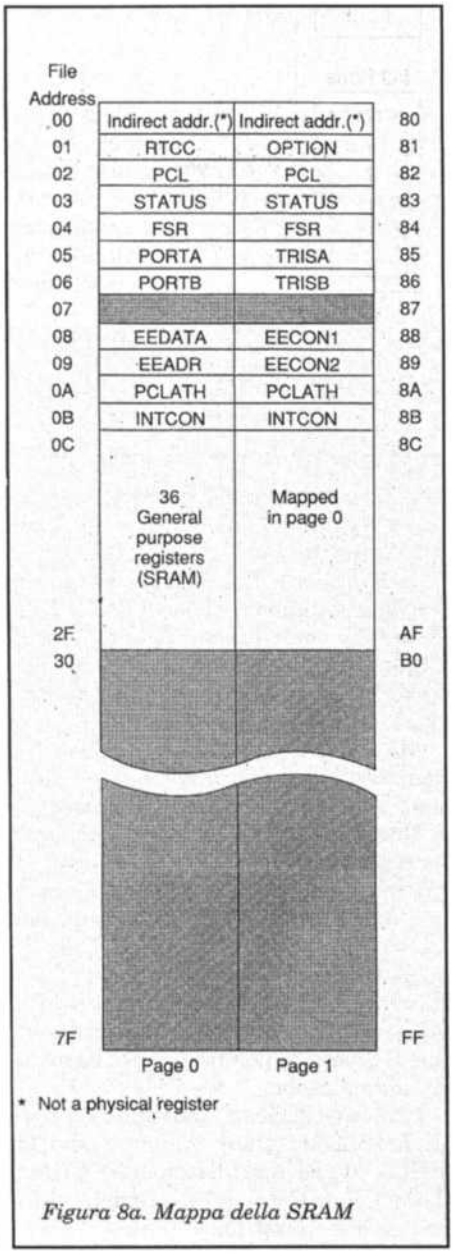


Figura 8a. Mappa della SRAM

I pin di connessione

Le due porte disponibili sul chip, hanno tre diversi interfacciamenti con i pin di I/O, a seconda delle funzioni che potrebbero svolgere.

In Figura 9a, possiamo vedere il più semplice, e cioè quello relativo alla porta A (RA0..RA3). Il flip-flop in alto a sinistra, serve per mantenere in memoria il valore del bit da presentare sul pin, mentre il flip-flop sotto di lui serve per selezionare il pin in input (three-state) oppure in output.

L'ultimo flip-flop esegue un latch sulla lettura del valore della tensione sul pin (ovviamente in digitale, cioè uno oppure zero).

In Figura 9b, invece, troviamo il diagramma a blocchi della porta B dal pin RB4 al pin RB7. Una differenza con la porta A consiste nel poter abilitare una resistenza di pull-up interna, senza quindi necessità di componenti esterni.

In più, e di maggior rilievo, abbiamo la generazione di un interrupt sul cambio di stato del valore di un pin.

Tanto per fare un esempio, se prendiamo un pulsante e lo colleghiamo al pin RB7, configurando tale pin come input e abilitando la resistenza di pull-up, è possibile avere un interrupt (mascherabile e se abilitato) ogni volta che il pulsante connette il pin RB7 alla massa.

I pin da RB0 a RB7 della porta B invece, sono connessi in modo molto similare alla porta A, come è visibile in Figura 9c. La differenza più rilevante è la presenza delle resistenze di pull-up.

Il reset del chip

Una attenzione particolare merita la sezione di reset del PIC16C84 il cui diagramma a blocchi è visibile in Figura 10.

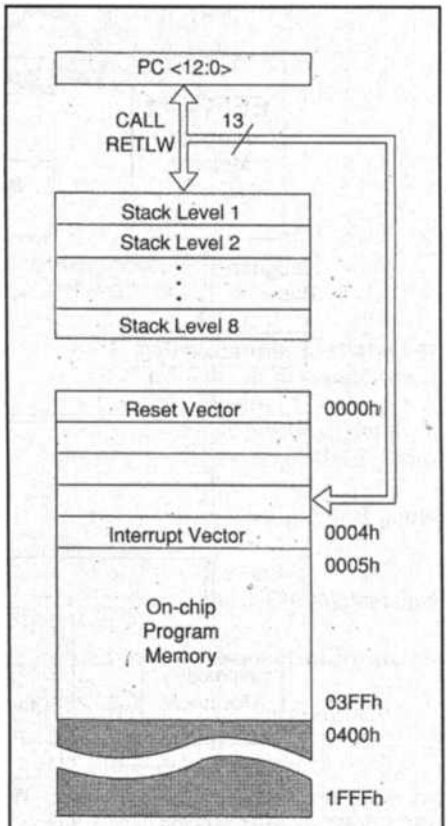


Figura 8b. Mappa della memoria di programma

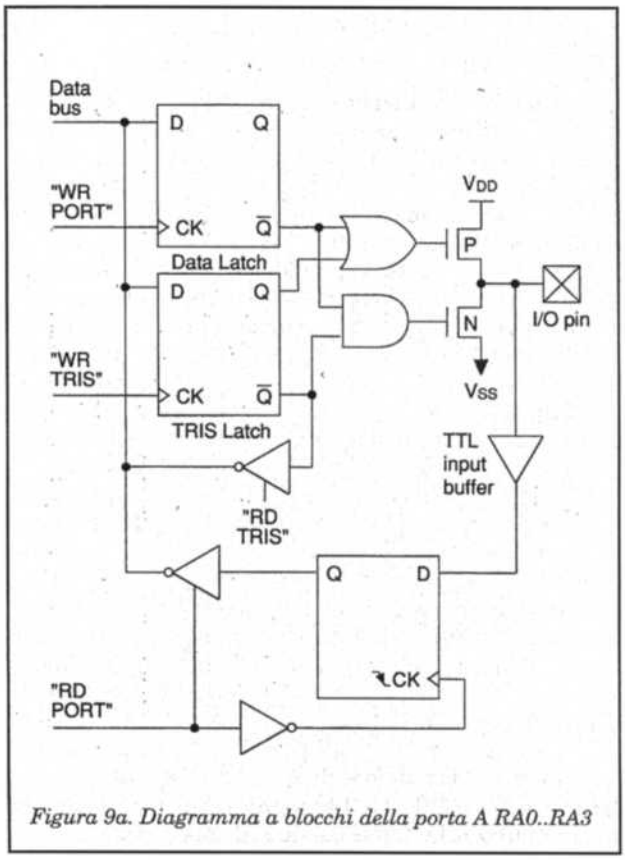


Figura 9a. Diagramma a blocchi della porta A RA0..RA3

Ci sono diversi modi per resettare il PIC, di cui il primo è la forzatura a massa del pin MCLR (Master CLeaR).

Quando questo pin viene connesso a massa, e per tutta la durata della connessione, il chip si trova in stato di reset.

Il secondo sistema per resettare il chip è l'impiego del watchdog. Poiché questo può essere abilitato o meno in fase di programmazione, il programmatore deve tenerne conto ed inserire l'istruzione CLRWDT prima dello scadere di circa 18 mS se non vuole correre il rischio di trovarsi il chip resettato.

Altro reset viene generato dal riconoscimento della tensione di alimentazione (alla prima accensione), assicurandoci che il chip partirà sempre dallo stato che noi abbiamo previsto.

In aggiunta a questo reset, abbiamo la possibilità di abilitare un altro reset all'accensione, di durata maggiore rispetto al precedente.

Questo meccanismo viene in genere impiegato quando il chip riceve l'alimentazione da un alimentatore collegato alla rete con un tempo di salita della tensione relativamente lungo.

Le istruzioni del PIC16C84

A questo punto ci sembra doveroso elencare le sole 37 (è un RISC!) istruzioni assembler del chip, comuni, tra l'altro, agli altri chip della famiglia PIC16Cxx.

Nella Tabella 2 le vediamo riassunte in tre gruppi principali: istruzioni orientate al byte, al bit e di controllo.

Velocemente vediamo cosa fanno, accennando prima alle notazioni che useremo: da adesso, il registro di lavoro sarà chiamato semplicemente "w", i vari registri saranno detti "f", ovvero "File register" e le costanti saranno definite "l", ovvero "Literal" e "k" ovvero "Kostant".

Il bit generico viene definito con "b".

Il registro di destinazione invece viene indicato con "d", ovvero "Destination".

Anche per chi non conosce la lingua inglese, dette queste tre cose, non ci saranno possibilità di errore.

Vediamo allora le istruzioni:

ADDWF f,d: Somma il valore del registro W al valore del registro f e mette il risultato in W se d=0, in f se d=1 (default).

ANDWF f,d: Esegue l'operazione booleana AND tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).

CLRF f: Pone a zero il valore del registro F.

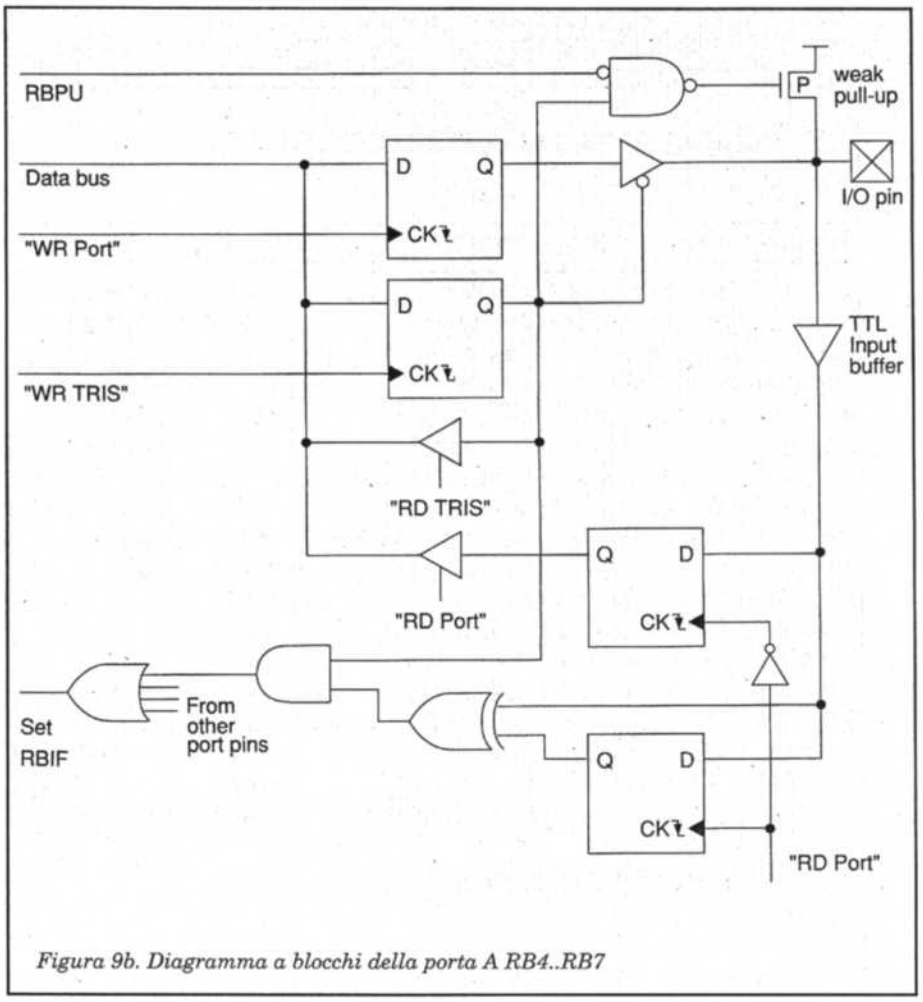


Figura 9b. Diagramma a blocchi della porta A RB4..RB7

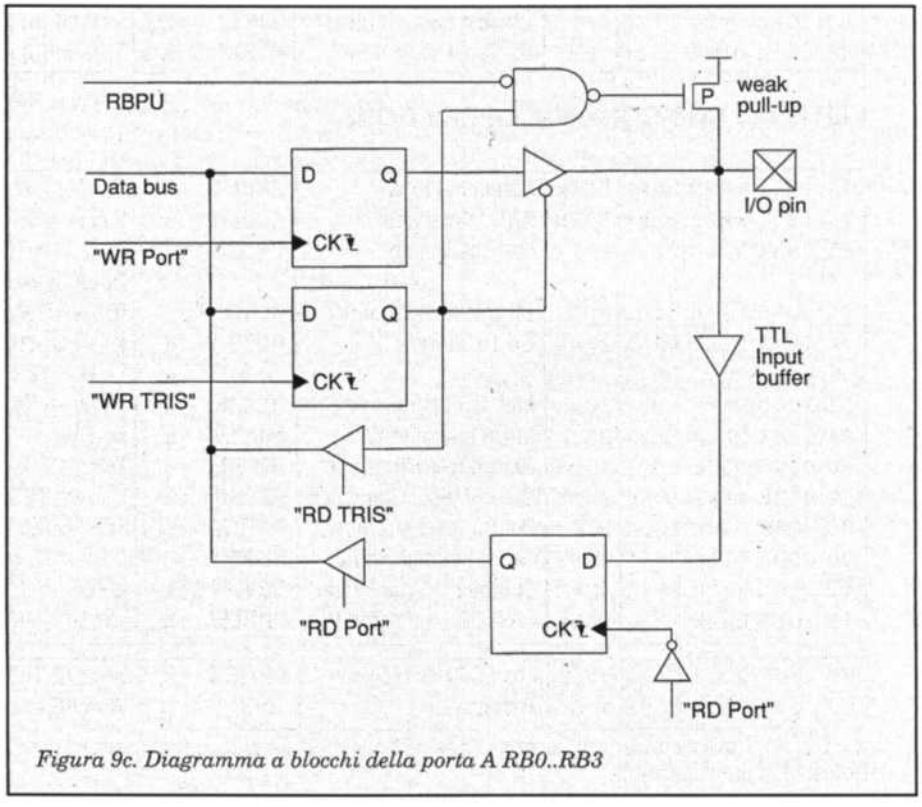


Figura 9c. Diagramma a blocchi della porta A RB0..RB3

Tabella 2. Sintesi delle istruzioni del PIC16C84

BYTE-ORIENTED FILE REGISTER OPERATIONS

13	8	7	6	0
OPCODE		d	f(FILE #)	

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Instruction-Binary	(Hex)	Name	Mnemonic, Operands		Operation	Status affected	Notes
00 0111 dfff ffff	07ff	Add W and f	**ADDWF**	f, d	W + f → d	C, DC, Z	2,3
00 0101 dfff ffff	05ff	AND W and f	**ANDWF**	f, d	W & f → d	Z	2,3
00 0001 1fff ffff	018f	Clear f	**CLRF**	f	0 → f	Z	3
00 0001 0XXX XXXX	0100	Clear W	**CLRW**	-	0 → W	Z	
00 1001 dfff ffff	09ff	Complement f	**COMF**	f, d	f̄ → d	Z	2,3
00 0011 dfff ffff	03ff	Decrement f	**DECF**	f, d	f -1 → d	Z	2,3
00 1011 dfff ffff	0Bff	Decrement f,Skip if Zero	**DECFSZ**	f, d	f - 1 → d, skip if zero	None	2,3
00 1010 dfff ffff	0Aff	Increment f	**INCF**	f, d	f + 1 → d	Z	2,3
00 1111 dfff ffff	0Fff	Increment f,Skip if zero	**INCFSZ**	f, d	f + 1 → d, skip if zero	None	2,3
00 0100 dfff ffff	04ff	Inclusive OR W and f	**IORWF**	f, d	W v f → d	Z	2,3
00 1000 dfff ffff	08ff	Move f	**MOVF**	f, d	f → d	Z	2,3
00 0000 1fff ffff	008f	Move W to f	**MOVWF**	f	W → f	None	3
00 0000 0XX0 0000	0000	No Operation	**NOP**	-	-	None	
00 1101 dfff ffff	0Dff	Rotate left f	**RLF**	f, d	f<n>→d<n+1>,C→d<0>,f<7> →C	C	2,3
00 1100 dfff ffff	0Cff	Rotate right f	**RRF**	f, d	f<n>→d<n-1>, C→d<7>, f<0>→C	C	2,3
00 0010 dfff ffff	02ff	Subtract W from f	**SUBWF**	f, d	f - W → d [f + W̄ + 1 → d]	C, DC, Z	2,3
00 1110 dfff ffff	0Eff	Swap halves f	**SWAPF**	f, d	f<0-3> ↔ f<4-7> → d	None	2,3
00 0110 dfff ffff	06ff	Exclusive OR W and f	**XORWF**	f, d	W ⊕ f → d	Z	2,3

BIT-ORIENTED FILE REGISTER OPERATIONS

13	10	9	7	6	0
OPCODE		b(BIT #)		f(FILE #)	

b = 3-bit bit address
f = 7-bit file register address

01 00bb bfff ffff	1bff	Bit Clear f	**BCF**	f, b	0 → f(b)	None	2,3
01 01bb bfff ffff	1bff	Bit Set f	**BSF**	f, b	1 → f(b)	None	2,3
01 10bb bfff ffff	1bff	Bit Test f,Skip if Clear	**BTFSC**	f, b	Test bit (b) in file (f): Skip if clear	None	
01 11bb bfff ffff	1bff	Bit Test f, Skip if Set	**BTFSS**	f, b	Test bit (b) in file (f): Skip if set	None	

LITERAL AND CONTROL OPERATIONS

13	8	7	0
OPCODE		k (LITERAL)	

k = 8-bit immediate value.

11 111X kkkk kkkk	3Ekk	Add literal to W	**ADDLW**	k	k + W → W	C,DC,Z	
11 1001 kkkk kkkk	39kk	AND Literal and W	**ANDLW**	k	k & W→ W	Z	
10 0kkk kkkk kkkk	2kkk	Call subroutine	**CALL**	k	PC + 1 → TOS, k → PC <10:0>, PCLATH <4:3> → PC <12:11>;	None	
00 0000 0110 0100	0064	Clear Watchdog timer	**CLRWDT**	-	0→WDT(and prescaler,if assigned)	TO, PD	
10 1kkk kkkk kkkk	2kkk	Go To address	**GOTO**	k	k → PC <10:0>, PCLATH <4:3> → PC <12:11>;	None	
11 1000 kkkk kkkk	38kk	Incl. OR Literal and W	**IORLW**	k	k v W → W	Z	
11 00XX kkkk kkkk	30kk	Move Literal to W	**MOVLW**	k	k → W	None	
00 0000 0000 1001	0009	Return from interrupt	**RETFIE**	-	TOS → PC, '1' → GIE	None	
11 01XX kkkk kkkk	34kk	Return, place literal in W	**RETLW**	k	k → W, TOS→ PC	None	
00 0000 0000 1000	0008	Return from subroutine	**RETURN**	-	TOS → PC	None	
00 0000 0110 0011	0063	Go into standby mode	**SLEEP**	-	0 → WDT, stop oscillator	TO, PD	
11 110X kkkk kkkk	3Ckk	Subtract W from literal	**SUBLW**	k	k - W → W	C,DC,Z	
11 1010 kkkk kkkk	3Akk	Excl. OR Literal and W	**XORLW**	k	k ⊕ W → W	Z	

00 0000 0110 0010	0062	Load OPTION register	**OPTION**	-	W → OPTION register	None	1
00 0000 0110 0fff	006f	Tristate port f	**TRIS**	f	W → I/O control register f	None	1

X = 0 or 1. The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all software tools.
Notes: See previous page

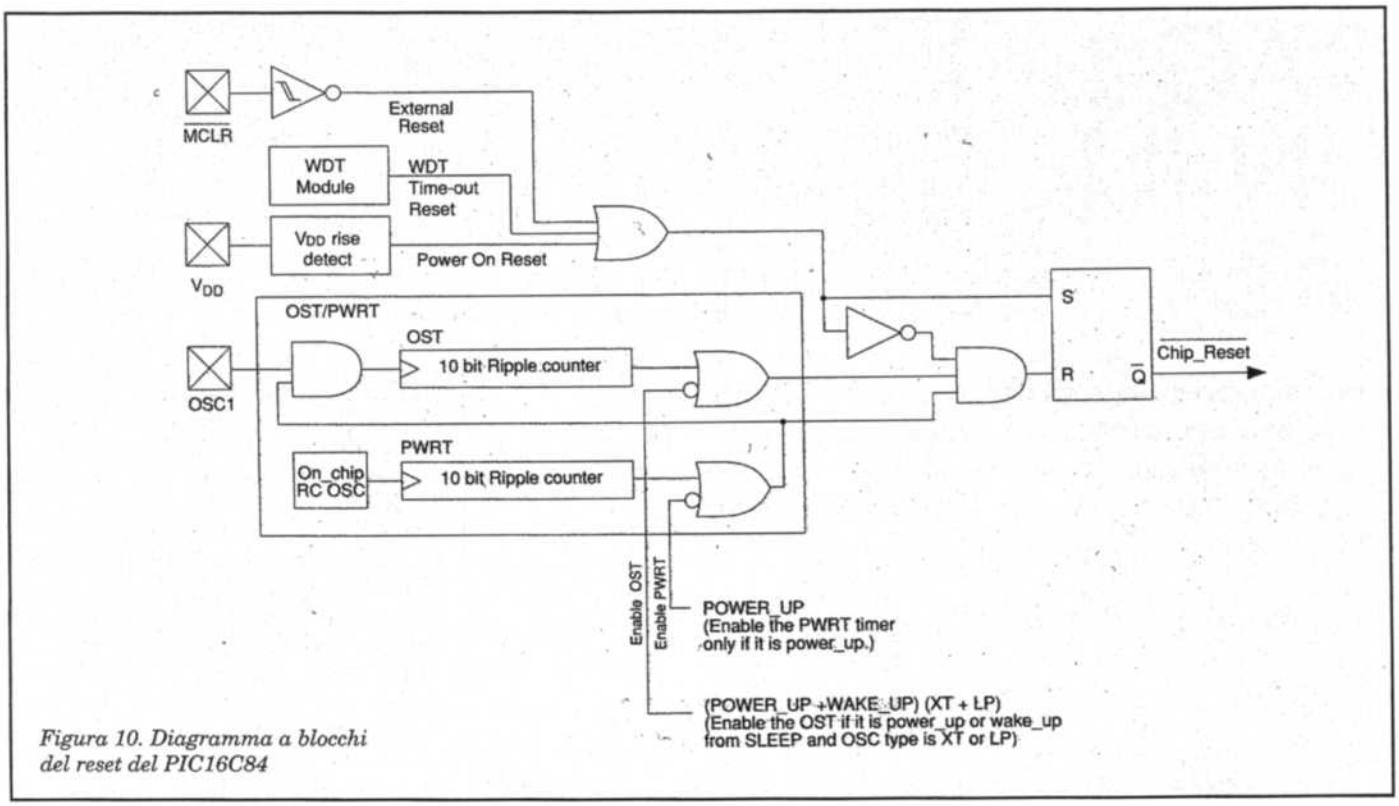


Figura 10. Diagramma a blocchi del reset del PIC16C84

CLRW: Pone a zero il valore del registro W.

COMF f,d: Complementa il valore del registro f (in pratica dove trova un "1" mette "0" e dove trova uno "0" mette un "1") e mette il risultato in W se d=0, in f se d=1 (default).

DECF f,d: Sottrae uno al registro f e mette il risultato in W se d=0, in f se d=1 (default).

DECFSZ f,d: Sottrae uno al registro f e, dopo, va a vedere se il valore del tale registro è divenuto zero. In caso affermativo, salta l'istruzione immediatamente successiva. Il risultato viene locato in W se d=0, in f se d=1 (default).

INCF f,d: Somma uno al registro f e mette il risultato in W se d=0, in f se d=1 (default).

INCFSZ f,d: Somma uno al registro f e, dopo, va a vedere se il valore del tale registro è divenuto zero. In caso affermativo, salta l'istruzione immediatamente successiva. Il risultato viene locato in W se d=0, in f se d=1 (default).

IORWF f,d: Esegue l'operazione booleana OR tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).

MOVF f,d: Legge il valore del registro f e lo copia in se stesso se d=1 (usato raramente) oppure in W se d=0.

MOVWF f: Legge il valore di W e lo copia nel registro f.

NOP: Non esegue alcuna operazione.

RLF f,d: Esegue un'operazione di shift a sinistra del registro f. Al bit 0 viene assegnato il valore del Carry, mentre il carry prende il valore del bit 7.

RRF f,d: Esegue un'operazione di shift a destra del registro f. Al bit 7 viene assegnato il valore del Carry, mentre il carry prende il valore del bit 0.

SUBWF f,d: Sottrae il valore del registro W al valore del registro f e mette il risultato in W se d=0, in f se d=1 (default).

SWAPF f,d: Scambia i due nibble del registro f e mette il risultato in W se d=0, in f se d=1 (default).

XORWF f,d: Esegue l'operazione booleana EX-OR tra il registro W ed il registro f e mette il risultato in W se d=0, in f se d=1 (default).

BCF f,b: Pone a zero il valore del bit b del registro f.

BSF f,b: Pone a uno il valore del bit b del registro f.

BTFSC f,b: Testa il valore del bit b del registro f: se tale valore è "0", salta l'istruzione immediatamente successiva.

BTFSS f,b: Testa il valore del bit b del registro f: se tale valore è "1", salta l'istruzione immediatamente successiva.

ADDLW k: Somma la costante k al valore del registro W.

ANDLW k: Esegue l'operazione booleana AND tra la costante k ed il registro W.

CALL k: Effettua una chiamata alla subroutine k.

CLRWDT: Pone a zero il registro del watchdog.

GOTO k: Esegue un salto alla label k.

IORLW k: Esegue l'operazione booleana OR tra la costante k ed il registro W.

MOVLW k: Pone nel registro w il valore della costante k.

RETFIE: Consente il ritorno dopo un interrupt.

RETLW k: Consente il ritorno dopo una CALL copiando il valore della costante k nel registro W.

RETURN: Consente il ritorno dopo una CALL.

SLEEP: Pone il controller in modalità SLEEP.

SUBLW k: Sottrae la costante k al valore del registro W.

XORLW k: Esegue l'operazione booleana EX-OR tra la costante k ed il registro W.

continua

Il software del programmatore di PIC viene fornito a fronte di un rimborso spese dalla Microlabs: 02/90967467

LA PROGRAMMAZIONE IN ASSEMBLER

Continuiamo il nostro corso completo che ci consentirà di programmare i chip della famiglia PIC. Ricordiamo che la base di partenza è rappresentata dal numero di gennaio in cui abbiamo pubblicato un semplice programmatore

Andrea Sbrana - 3ª parte
su gentile concessione della Microlabs

Da questo mese, iniziamo a vedere in dettaglio le varie fasi dell'implementazione software dei circuiti che vogliamo realizzare, partendo da esempi molto banali, ma in grado di far capire il meccanismo della programmazione assembler.

Il primo argomento che tratteremo è la prima operazione da effettuare quando si programma un microcontrollore, e cioè settare correttamente le porte ed inizializzare i registri dedicati che poi andremo ad utilizzare.

Ma prima di far ciò, ricordiamo che l'assemblatore impiegato è l'MPASM, fornito gratuitamente dalla Microchip e copiabile senza incorrere in sanzioni penali.

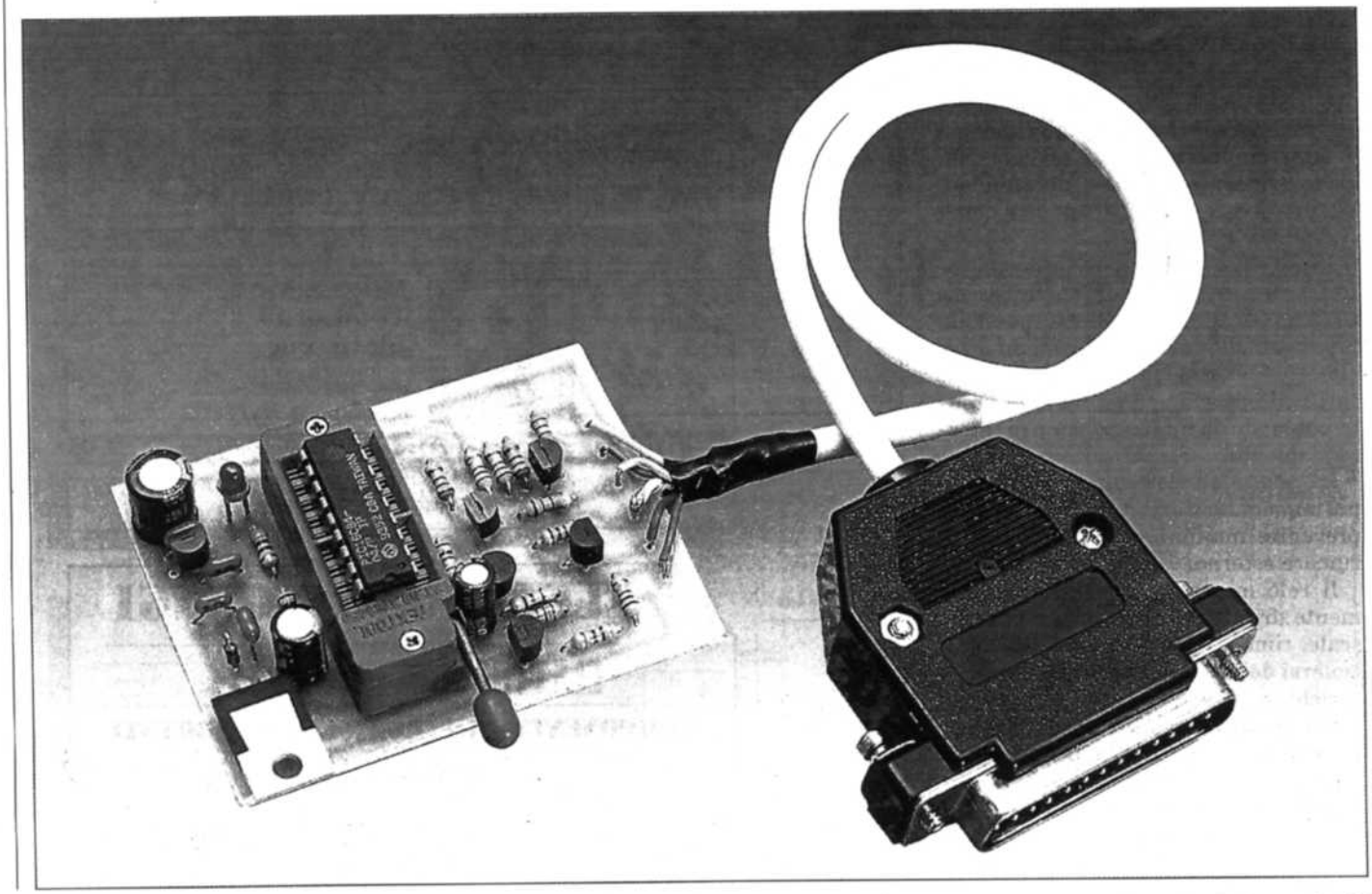
A tale proposito ricordiamo che è possibile accedere a tale eseguibile sia tramite BBS che tramite internet.

Telefonando alla Eurelettronica al numero 02/45784234 avrete le informazioni necessarie per fare queste operazioni.

In Figura 11 troviamo il circuito che serve per eseguire le prove di questo mese e ricordiamo che la scheda già pronta è stata presentata sul numero di dicembre '94 di Progetto e che potrà essere richiesta direttamente alla Eurelettronica.

In caso contrario potrete benissimo approntare una comune millefori, data l'estrema semplicità della realizzazione.

Con il programma 1 vedremo inoltre alcune direttive proprie dell'assemblatore. Analizziamo allora il programma PROG01.



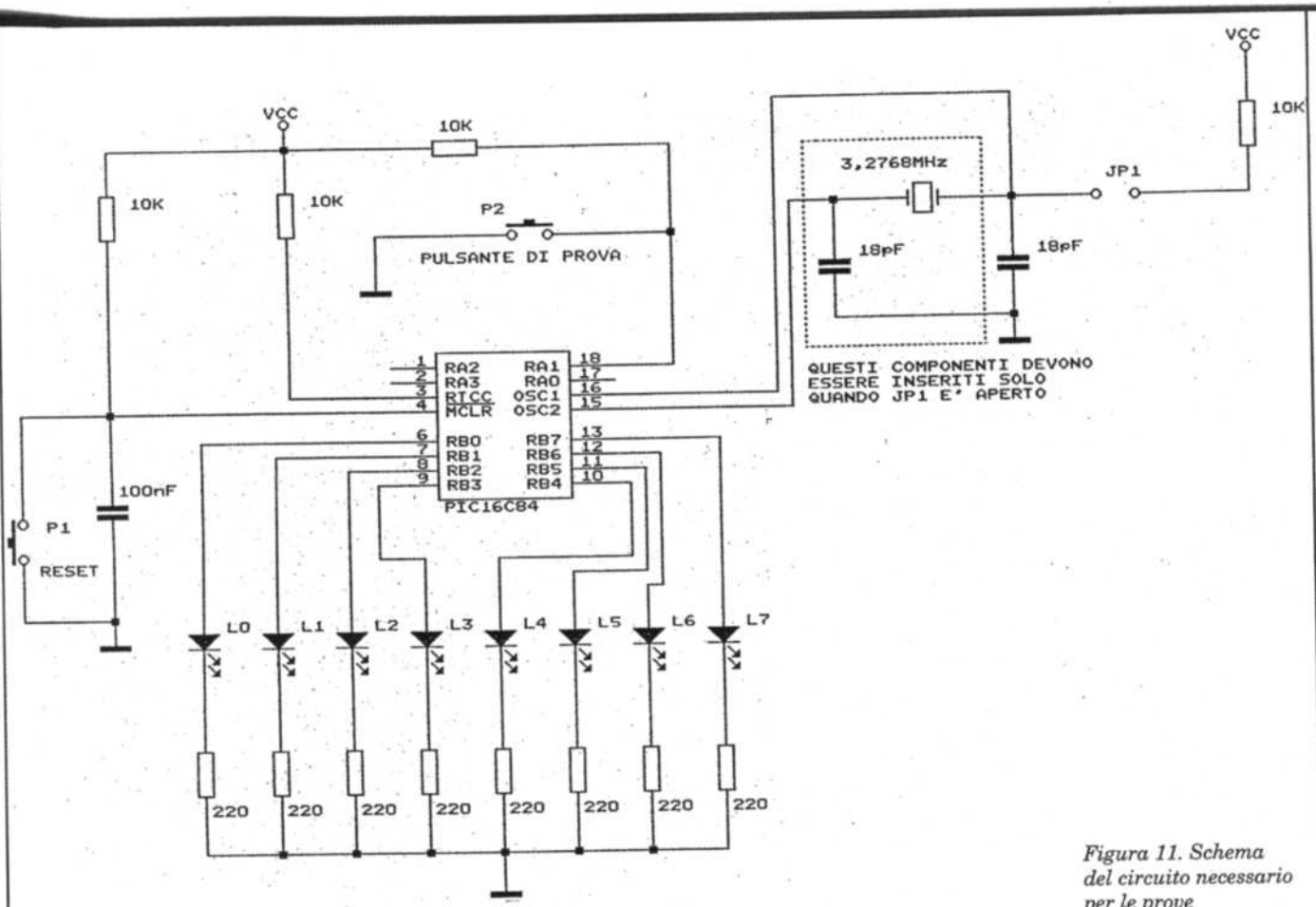
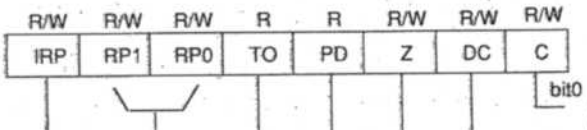


Figura 11. Schema del circuito necessario per le prove



ADDRESS: 03h
RESET CONDITION: 000??XXX
TO, PD are set or reset as shown in Table 3.9.2.1

CARRY/$\overline{\text{BORROW}}$ BIT:
For ADDWF, SUBWF , ADDLW and SUBLW instructions, this bit is set if there is a carry out from the most significant bit of the resultant. Note that a subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

DIGIT CARRY/$\overline{\text{BORROW}}$ BIT:
For ADDWF, SUBWF , ADDLW and SUBLW instructions, this bit is set if there is a carry out from the 4th low order bit of the resultant.

ZERO BIT:
Set if the result of an arithmetic or logic operation is zero. Reset otherwise. Additionally, MOVF instruction will affect the Z bit.

POWER DOWN BIT:
Set to "1" during power up or by a CLRWDT command. This bit is cleared to "0" by a SLEEP instruction.

TIME-OUT BIT:
Set to "1" during power up and by the CLRWDT and SLEEP command. This bit is cleared to "0" by a watchdog timer time out.

REGISTER PAGE SELECT BITS FOR DIRECT ADDRESSING:

RP1,0:
00 : page 0 (00h - 7Fh)
01 : page 1 (80h - FFh)
10 : page 2 (100h - 17Fh)
11 : page 3 (180h - 1FFh)

Each page is 128 bytes.
Only RP0 is useful in PIC16C84. Bit RP1 can be used as a general purpose read/write bit. However, this may affect upward compatibility with future products.

REGISTER PAGE SELECT BITS FOR INDIRECT ADDRESSING:

IRP0:
0 : page 0,1 (00h - FFh)
1 : page 2,3 (100h - 1FFh)

This bit is effectively not used in the PIC16C84.
It may be used as a general purpose read/write bit. However, this may affect upward compatibility with future product.

Figura 12. Significato dei bit del registro di stato

La prima riga ovviamente serve per ricordarci il titolo del programma che andiamo a scrivere.

La seconda invece fornisce due direttive all'assemblatore: la prima indica che il formato del file di uscita per il programmatore è di tipo INHX8S, la seconda che il PIC da programmare è del tipo 16C84.

Le sei righe successive sono associazioni di label (etichette) ai registri: se dobbiamo richiamare il registro di stato, è molto più facile ricordarsi "STAT" che non il suo indirizzo esadecimale (03h).

La riga che inizia con il cancelletto, assegna alla label LED0 il pin 0 della porta B. In questo modo, cambiando in seguito l'hardware, sarà molto facile riassegnare i pin di I/O.

ORG 0 è una direttiva che fa in modo che la istruzione successiva sia posta all'indirizzo 0 della memoria di programma.

Nel nostro caso si esegue un "goto START", cioè la prossima istruzione del programma sarà la "bsf STAT,5".

Questa istruzione, pone a 1 il bit numero 5 dello status register. In Figura 12 possiamo vedere che in questo modo viene selezionato il banco 1 della SRAM.

Con l'istruzione successiva, viene caricato in W il valore binario "0100".

L'istruzione "movwf" poi, copia tale valore nel registro TRISA.

In questo modo si definiscono le direzioni dei pin della porta A: dove abbiamo messo uno "0", il pin corrispondente sarà configurato come uscita, dove abbiamo messo un "1", il pin corrispondente sarà configurato come ingresso.

Lo stesso dicasi per le istruzioni relative alla porta B.

Con l'istruzione "clrf INTCON" si disabilitano tutte le sorgenti di interrupt del PIC.

Si ripassa poi al banco 0 della SRAM ponendo a 0 il bit 5 del registro di stato.

Con le due istruzioni successive, si pongono a zero tutti i pin di uscita, quindi, ripassando alla Figura 11, i led saranno tutti spenti.

Poiché il programma è terminato, per vedere se effettivamente ha funzionato, abbiamo inserito l'istruzione "bsf LED0", che farà accendere il solo led 0. Le ultime due righe sanciscono la fine del programma.

Ovviamente, questo non è un programma vero e proprio, però tramite questo banale esempio sarete in grado di inizializzare i PIC16C84 in funzione del vostro hardware.

In Tabella 3 troviamo lo stato di tutti i registri del PIC16C84 dopo un reset. I valori dei registri general pourpose rimangono inalterati.

Un primo controllo di tempo

Vediamo ora come implementare un controllo di tempo rudimentale con il programma PROG02 (programma 2).

Con PROG02, dopo aver dato l'alimentazione, vedremo accendere il led 2 dopo un tempo di circa 5 secondi, ovviamente impiegando un quarzo da 3,579545 MHz.

Sono state introdotte nuove label, corrispondenti ai quattro registri impiegati e al led numero 1.

Prendiamo in esame la subroutine DELA2M. Poich0 sappiamo che il PIC divide per 4 la frequenza di clock, se gli applichiamo un oscillatore da 3,579545 MHz, ogni istruzione verrà eseguita in 1,117 microsecondi.

Quindi, per ottenere un millisecondo saranno necessarie circa 888 istruzioni. Il ciclo più interno di questa subroutine (relativo a DEL1) esegue esattamente 888 istruzioni, prima di passare al ciclo più esterno (DEL0) che conterà 20 cicli da 1mS per ottenere i 20 mS richiesti.

Andando ora a vedere il main program, dopo aver acceso il led 0, si creano due cicli con cui implementare il ritardo richiesto: con RIT2 si realizzano ritardi da 1 secondo, perché la subroutine da 20 mS viene richiamata per 50 volte (20mS x 50 = 1.000 mS), mentre con RIT1 vengono contati i secondi di attesa veri e propri. Modificando i valori caricati nei registri, è possibile modificare tutte le tempistiche.

Al termine dei 5 secondi, si accenderà il led 1.

Questo modo di calcolare tempi può andar bene se non abbiamo particolari esigenze di precisione, ma ci accorgiamo che per piccoli errori di alcuni microsecondi ad ogni ciclo, su alcuni secondi possiamo trovare diversi millisecondi di scarto e, su minuti, troveremmo sicuramente secondi di differenza.

In più, durante queste attese, il chip non può eseguire altri lavori, impegnato come è nel contare il tempo!

Allora si impone un altro modo di lavorare, e precisamente impiegando un registro, l'RTCC (Real Time Clock Counter) che viene incrementato automaticamente ogni n istruzioni eseguite.

Il valore di questo n dipende da un prescaler che può essere abilitato o meno.

PROGRAMMA 1

```
TITLE 'PROG01: Prova inizializzazione porte per Progetto'
list F=INHX8M,P=16C84
;-----------------------------------------------------------
STAT   EQU 03H    ; Registro di stato
PORTA  EQU 05H    ; Porta A
PORTB  EQU 06H    ; Porta B
INTCON EQU 0BH    ; Registro abilitazione interrupt
TR_A   EQU 85H    ; Tris A
TR_B   EQU 86H    ; Tris B
;-----------------------------------------------------------
#define LED0        PORTB,0       ; Led 0 sulla porta B pin 0
;-----------------------------------------------------------
    ORG   0
    goto  START ;Reset vector
    nop
    nop
    nop
    goto  INTERP          ;Interrupt vector
;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START  bsf  STAT,5 ; Seleziona SRAM banco 1
    movlw  b'0010' ; RA0,RA2,RA3 out   RA1 in
    movwf  TR_A    ;
    movlw  b'00000000'  ; RB0...RB7 out
    movwf  TR_B    ;
    clrf   INTCON       ; Disabilita interrupt
    bcf    STAT,5 ; Seleziona SRAM banco 0
    clrf   PORTA; Uscite della porta A tutte a 0
    clrf   PORTB; Uscite della porta B tutte a 0
    bsf    LED0   ; Accensione led 0
;-------------------- main program ----------------------
MAIN
    clrwdt  ; Corpo del programma
    ; .
    ; .
    goto   MAIN   ; .
;-----------------------------------------------------------
INTERP    ; Vettore interruzioni
    retfie    ;
;-----------------------------------------------------------
    END
```

PROGRAMMA 2

```
TITLE 'PROG02: Prova temporizzazione 1 per Progetto'
list F=INHX8M,P=16C84
;-----------------------------------------------------------
STAT   EQU 03H    ; Registro di stato
PORTA  EQU 05H    ; Porta A
PORTB  EQU 06H    ; Porta B
INTCON EQU 0BH    ; Registro abilitazione interrupt
TR_A   EQU 85H    ; Tris A
TR_B   EQU 86H    ; Tris B
;-----------------------------------------------------------
FR01   EQU 0CH    ; Ritardo 20mS
FR02   EQU 0DH    ; Ritardo 20mS
FR03   EQU 0EH    ; Ritardo 1S
FR04   EQU 0FH    ; Ritardo N secondi
;-----------------------------------------------------------
#define LED0       PORTB,0      ; Led 0 sulla porta B pin 0
#define LED1       PORTB,1      ; Led 1 sulla porta B pin 1
;-----------------------------------------------------------
     ORG   0
     goto  START ;Reset vector
     nop
     nop
     nop
     goto  INTERP        ;Interrupt vector
;-------------- subroutine RITARDO 20 mS --------------------
; con frequenza 3,579545 MHz
;-----------------------------------------------------------
DELA2M movlw   .20     ; Carica 20 in W
     movwf  FR01   ; Copia W in FR01
     movlw  .177   ; 888 operazioni a 1,117 uS cd.
DEL0 movwf     FR02   ; Copia W in FR02
DEL1 clrwdt    ; Azzera watchdog
     nop    ; Nessuna operazione
     decfsz   FR02 ; Decrementa FR02, salta se 0
     goto   DEL1   ; Vai a DEL1
     decfsz   FR01 ; Decrementa FR01, salta se 0
     goto   DEL0   ; Vai a DEL0
     return ; Ritorna dopo la CALL
;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START  bsf  STAT,5 ; Seleziona SRAM banco 1
     movlw  b'0010' ; RA0,RA2,RA3 out   RA1 in
     movwf  TR_A   ;
     movlw  b'00000000'  ; RB0...RB7 out
     movwf  TR_B   ;
     clrf   INTCON       ; Disabilita interrupt
     bcf    STAT,5 ; Seleziona SRAM banco 0
     clrf   PORTA; Uscite della porta A tutte a 0
     clrf   PORTB; Uscite della porta B tutte a 0
     bsf    LED0   ; Accensione led 0
;-------------------- main program ------------------------
     movlw  .5     ; Carica 5 in W
     movwf  FR04   ; Copia W in FR04
RIT1 movlw          .50     ; Carica 50 in W
     movwf  FR03   ; Copia W in FR03
RIT2 call   DELA2M        ; Vai alla subroutine DELA2M
     decfsz   FR03 ; Decrementa FR03, salta se 0
     goto   RIT2   ; Vai a RIT2
     decfsz   FR04 ; Decrementa FR04, salta se 0
     goto   RIT2   ; Vai a RIT1
     bsf    LED1   ; Accensione led 1
MAIN       ; Corpo del programma
     clrwdt  ;
     ;
     goto   MAIN   ;
;-----------------------------------------------------------
INTERP     ; Vettore interruzioni
     retfie  ;
;-----------------------------------------------------------
     END
```

PROGRAMMA 3

```
TITLE 'PROG03: Prova temporizzazione 2 per Progetto'
list F=INHX8M,P=16C84
;-----------------------------------------------------------
RTCC   EQU 01H    ; Real Time Clock Counter
STAT   EQU 03H    ; Registro di stato
PORTA  EQU 05H    ; Porta A
PORTB  EQU 06H    ; Porta B
INTCON EQU 0BH    ; Registro abilitazione interrupt
TR_A   EQU 85H    ; Tris A
TR_B   EQU 86H    ; Tris B
OPTIO  EQU 81H    ; Registro OPTION
;-----------------------------------------------------------
FR01   EQU 0CH    ; Ritardo 20mS
FR02   EQU 0DH    ; Ritardo 20mS
FR03   EQU 0EH    ; Ritardo 1S
FR04   EQU 0FH    ; Ritardo N secondi
;-----------------------------------------------------------
#define LED0       PORTB,0      ; Led 0 sulla porta B pin 0
#define LED1       PORTB,1      ; Led 1 sulla porta B pin 1
;-----------------------------------------------------------
     ORG   0
     goto  START ;Reset vector
     nop
     nop
     nop
     goto  INTERP        ;Interrupt vector
;-------------- subroutine RITARDO 20 mS -------------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------
DELA2M clrwdt     ;
     movf   RTCC,0       ; Controllo se finiti 20mS
     SKPZ   ;
     goto   DELA2M       ;
     movlw  .128    ;
     movwf  RTCC    ;
     return ; Ritorna dopo la CALL
;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START  bsf  STAT,5 ; Seleziona SRAM banco 1
     movlw  b'0010' ; RA0,RA2,RA3 out   RA1 in
     movwf  TR_A   ;
     movlw  b'00000000'  ; RB0...RB7 out
     movwf  TR_B   ;
     clrf   INTCON       ; Disabilita interrupt
     movlw  b'11000110'  ; Prescaler 1:128
     movwf  OPTIO ; Copia W in OPTION
     bcf    STAT,5 ; Seleziona SRAM banco 0
     clrf   PORTA; Uscite della porta A tutte a 0
     clrf   PORTB; Uscite della porta B tutte a 0
     bsf    LED0   ; Accensione led 0
;-------------------- main program ------------------------
     movlw  .128    ; Settaggio iniziale RTCC
     movwf  RTCC    ;
     movlw  .5      ; Carica 5 in W
     movwf  FR04    ; Copia W in FR04
RIT1 movlw          .50     ; Carica 50 in W
     movwf  FR03   ; Copia W in FR03
RIT2 call   DELA2M        ; Vai alla subroutine DELA2M
     decfsz   FR03 ; Decrementa FR03, salta se 0
     goto   RIT2   ; Vai a RIT2
     decfsz   FR04 ; Decrementa FR04, salta se 0
     goto   RIT1   ; Vai a RIT1
     bsf    LED1   ; Accensione led 1
MAIN
     clrwdt  ;
     ;
     goto   MAIN   ;
;-----------------------------------------------------------
INTERP     ; Vettore interruzioni
     retfie  ;
;-----------------------------------------------------------
     END
```

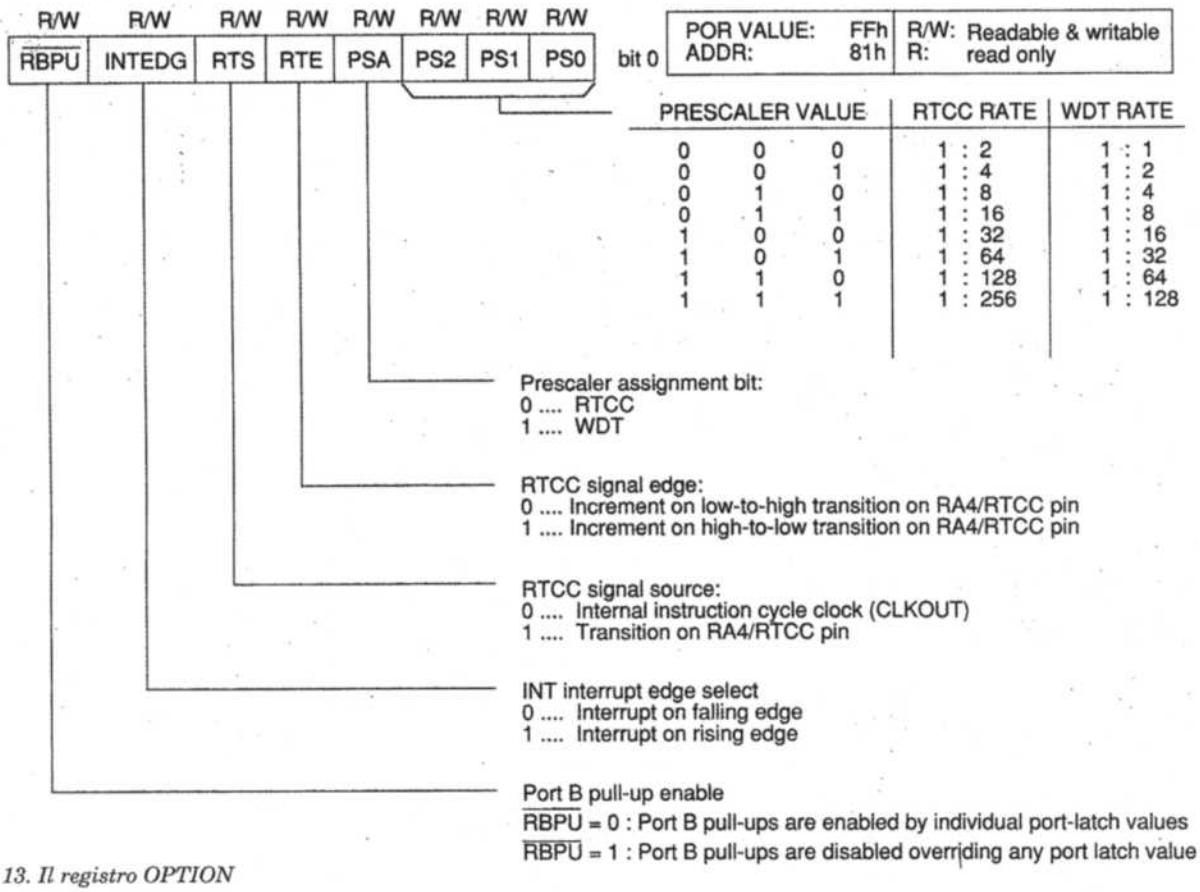


Figura 13. Il registro OPTION

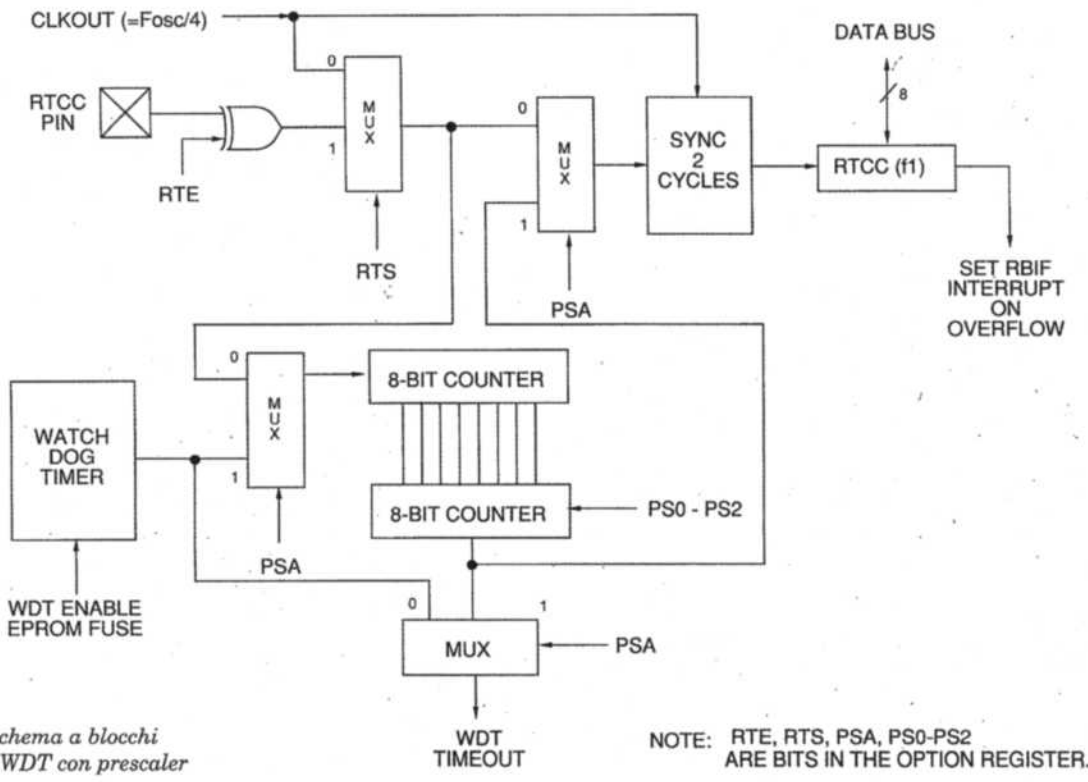


Figura 14. Schema a blocchi dell'RTCC e WDT con prescaler

Tabella 3. Settaggio dei registri dopo un reset

Register	Address	Power-on reset (POR)	WDT time-out reset during normal operation	WDT time-out reset during SLEEP	MCLR reset during normal operation	MCLR reset during SLEEP	Wake-up through interrupt
W	–	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
INDIR	00h	–	–	–	–	–	–
RTCC	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PC	02h	0000h	0000h	PC + 1	0000h	0000h	PC + 1
STATUS	03h	0001 1xxx	0000 1 uuu	uuu0 0uuu	000u uuuu	0001 0uuu	uuu1 0uuu
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT A	05h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT B	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TRIS A	85h	- - -1 1111	- - -1 1111	- - -u uuuu	- - -1 1111	- - -1 1111	- - -u uuuu
TRIS B	86h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111	uuuu uuuu
OPTION	81h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EECON1	88h	--- 0 0000	---0 ?000†	---u uuuu	---0 ?000†	---0 ?000†	---u uuuu
EECON2	89h	–	–	–	–	–	–
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu	0000 000u	0000 0000	uuuu uuuu*

Legend: – = unimplemented, reads as '0' u = unchanged x = unknown

** In the event of wake-up through interrupt, one or more of the interrupt flags will be set. Other bits in INTCON will remain unchanged.*

† WRERR (bit3) will be set if reset occurrend during EEPROM write.

In Figura 13 vediamo come si deve configurare il registro OPTION, cioè il registro che definisce i parametri dell'RTCC e del prescaler. Il bit 7 ed il bit 6 settano altre due funzioni, e cioè l'abilitazione delle resistenze di pull-up sulla porta B e la selezione dell'interrupt sul fronte di salita oppure sul fronte di discesa. Il bit 5 indica se l'incremento dell'RTCC deve avvenire per transizione di livello sul pin RTCC, oppure in relazione al ciclo di clock interno. Il bit 4 setta se l'incremento deve avvenire su fronte di salita oppure di discesa. Il bit 3 invece assegna il prescaler al watchdog oppure all'RTCC.

Con i bit 2, 1 e 0, si imposta il valore di divisione del prescaler secondo la tabellina riportata a fianco.

In Figura 14 è possibile vedere il diagramma a blocchi della condivisione del prescaler tra watchdog e RTCC.

Vediamo allora come implementare dei ritardi precisi avvalendosi dell'RTCC, seguendo il programma PROG03 (programma 3).

La subroutine di attesa fine 20 mS è stata completamente sostituita da un'altra che si avvale del conteggio interno dell'RTCC. Vediamo che il registro RTCC deve essere caricato con un valore, 128, e che il prescaler deve dividere per 128.

Questa combinazione è del tutto casuale, perché potremmo avere dei valori diversi con altri quarzi o con altre tempistiche. Ad esempio, se volessimo con la stessa subroutine ottenere 5mS, sarebbe sufficiente settare il prescaler per una divisione di 32.

Come calcolare questi valori? Se abbiamo il quarzo da 3,2768MHz, la frequenza di lavoro sarà di 3.276.800 Hz/4 = 819.200 Hz.

Supponendo di prescalerizzare tale frequenza con rapporto 1:32, allora per ottenere 5mS dovremo applicare la formula:

(256 - 128) x (1/(819.200/32)) = 5 mS

Con un quarzo da 4.096 MHz invece dovremo caricare l'RTCC con il numero 96.

Da notare le due righe che sono state aggiunte per settare il registro OPTION con i valori da noi richiesti.

In questo modo otterremo dei tempi precisi al microsecondo, tanto che le applicazioni principali sono proprio timer, orologi e contasecondi.

Per esercitazione, consigliamo di modificare il programma PROG03 per far accendere il led 1 ogni secondo.

Nella prossima puntata troverete una delle possibili soluzioni.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Il nostro corso di programmazione sta entrando nel vivo, grazie alla serie di esempi che abbiamo incominciato a proporre a partire dallo scorso numero della rivista. Vediamo ora come, con qualche semplice modifica, è possibile ottenere una sequenza di istruzioni più complesse

Andrea Sbrana - 4ª parte
su gentile concessione della Microlabs

Poiché lo scorso mese ci siamo lasciati con un piccolo compito da svolgere, vediamo subito come potevamo implementare un Led lampeggiante con frequenza di 2 Hz, ovvero un secondo acceso ed un secondo spento. Il circuito elettrico è lo stesso dello scorso mese, mentre il programma è il PROG04.

Si nota subito che le sole due istruzioni in più da inserire rispetto al programma PROG03.asm, sono:

movlw b'00000001' e
xorwf PORTB.

Vediamo perché con queste due istruzioni si riesce a far accendere e spegnere alternativamente un Led: l'operazione booleana OR-ESCLUSIVO, dati due bit in ingresso, restituisce un bit che vale zero se i due ingressi sono uguali, uno se i due ingressi sono diversi.

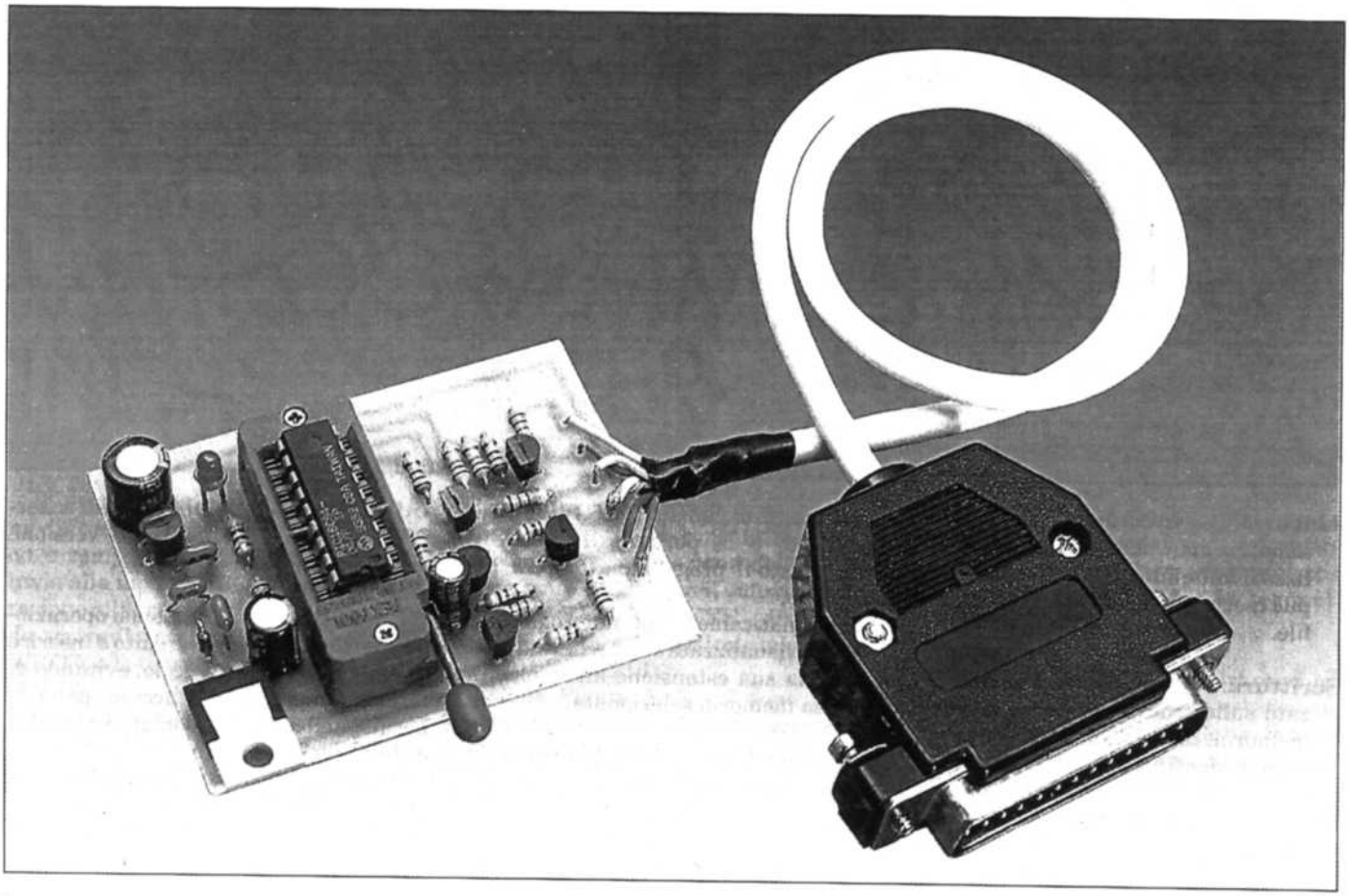
Nel registro W era stato caricato il numero binario 00000001. Il secondo registro su cui operare era la porta B.

Ora, qualsiasi valore fosse contenuto nei bit da 7 a 1 della porta B, con lo XOR con W lo avremmo ritrovato dopo l'operazione stessa, poiché lo XOR di 0 con 0 dà 0 e lo XOR di 1 con 0 dà 1.

Ma al bit 0 viene eseguito lo XOR con un 1, quindi se tale bit era 1, dopo lo XOR il suo valore sarà 0 e viceversa.

Allo stesso modo era possibile, con le solite due istruzioni, far lampeggiare tutti i Led alternativamente, semplicemente modificando il valore caricato in W.

Per far lampeggiare il Led a frequenza doppia invece, è sufficiente modificare il



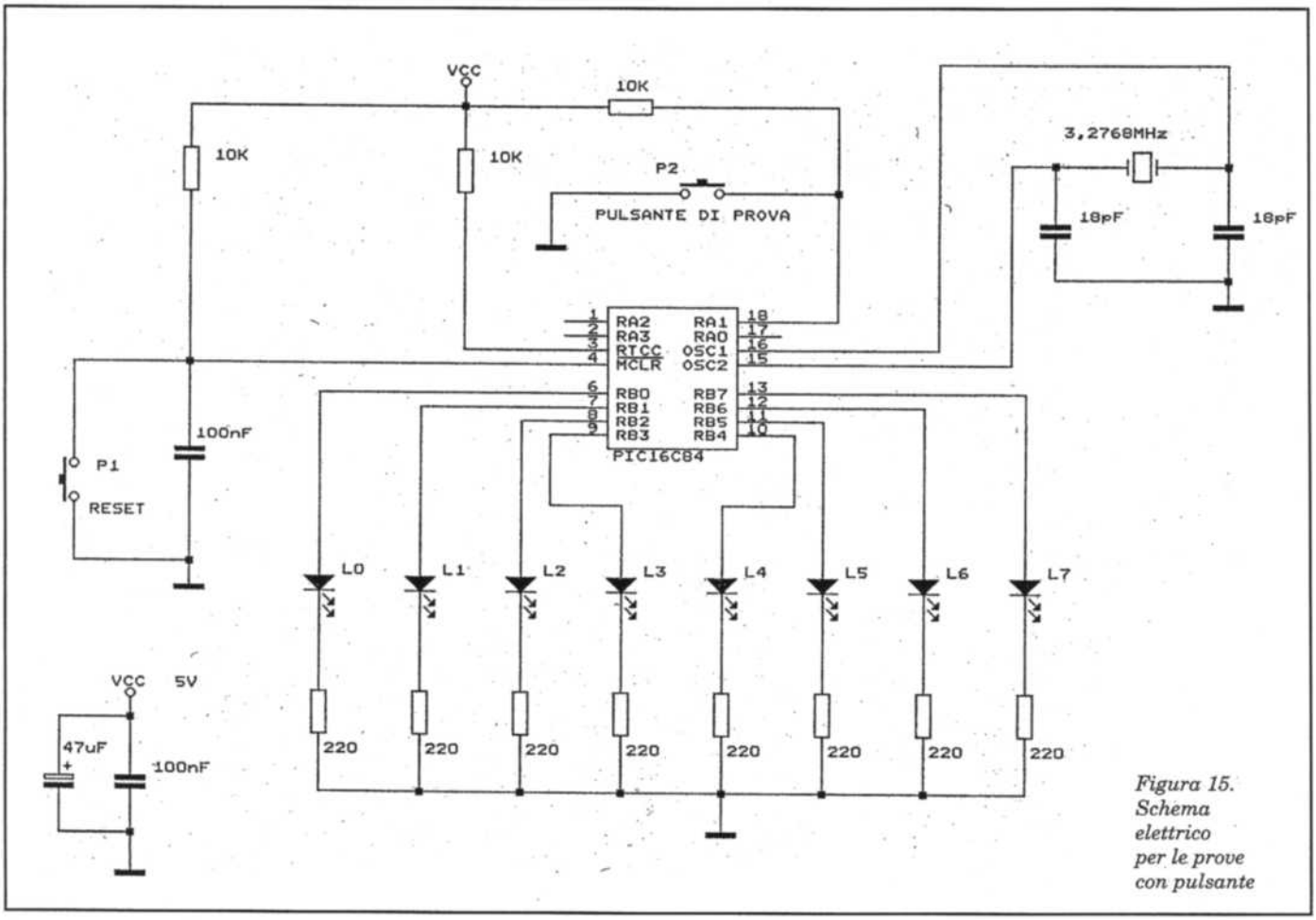


Figura 15. Schema elettrico per le prove con pulsante

valore con cui viene caricato il registro FR03 da 50 in 25, oppure modificare il valore del prescaler da 1:128 a 1:64.

Dopo aver visto in modo abbastanza preciso come controllare le tempistiche, vediamo ora come reagire a stimoli esterni quali la pressione di un pulsante.

Interfacciamento con pulsanti

Prendendo in esame lo schema elettrico di Figura 15, notiamo che il pulsante P2 è connesso sul pin RA1 ed ha una resistenza di pull-up da 10 kΩ.

Questo significa che il controller, quando va a leggere il valore del bit 1 sulla porta A, trova un "1" se il pulsante non è premuto, mentre trova uno "0" se il pulsante è premuto. Potevamo anche scegliere come ingresso la porta B, che ha già le resistenze di pull-up interne, ma in questo modo potete sfruttare il circuito già realizzato il mese scorso.

Quali sono le problematiche di un ingresso a pulsante? A prima vista nessuna, ma ci accorgeremo che, nella pratica, è possibile trovarsi di fronte a molti inconvenienti, tutti chiaramente gestibili via software.

Il primo di questi è il classico impulso di rumore che dura per pochi microsecondi, ma che potrebbe essere interpretato come la pressione del pulsante.

Facciamo presente che noi andremo a testare il pin di ingresso pulsante con l'istruzione BTFSS oppure con l'istruzione BTFSC e che, in funzione del valore letto, stabiliremo se il pulsante è stato premuto o meno.

Il secondo problema che può sorgere è il classico "rimbalzo", ovvero l'onda sinusoidale che viene generata dalla pressione del rimbalzo, per cui è possibile vedere il pulsante prima premuto e, dopo soli pochi millisecondi, lo stesso pulsante rilasciato, mentre invece non lo è.

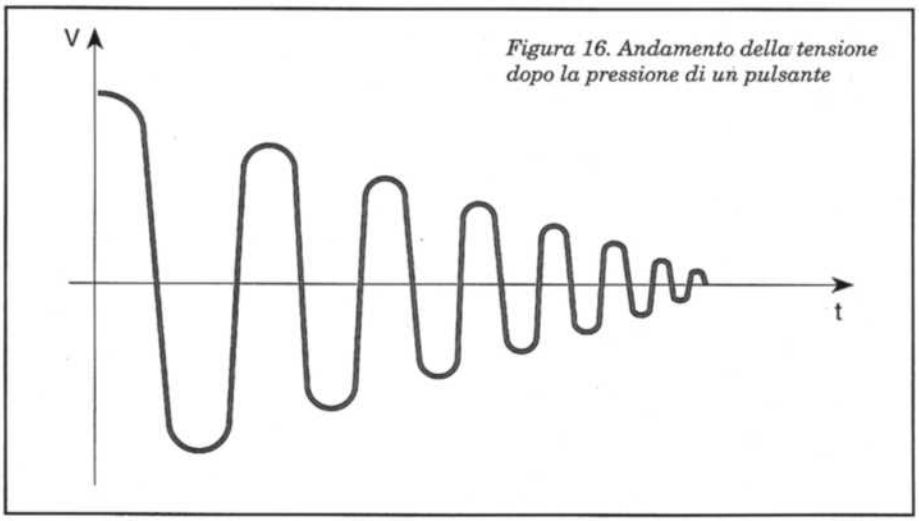


Figura 16. Andamento della tensione dopo la pressione di un pulsante

PROGRAMMA 4

```
TITLE 'PROG04: Led lampeggiante per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H          ; Real Time Clock Counter
STAT    EQU 03H          ; Registro di stato
PORTA   EQU 05H          ; Porta A
PORTB   EQU 06H          ; Porta B
INTCON  EQU 0BH          ; Registro abilitazione interrupt
TR_A    EQU 85H          ; Tris A
TR_B    EQU 86H          ; Tris B
OPTIO   EQU 81H          ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH          ; Ritardo 20mS
FR02    EQU 0DH          ; Ritardo 20mS
FR03    EQU 0EH          ; Ritardo 1S
;-----------------------------------------------------------------
        ORG   0
        goto  START      ;Reset vector
        nop
        nop
        nop
        goto  INTERP     ;Interrupt vector
;---------------- subroutine RITARDO 20 mS ----------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------------
DELA2M clrwdt            ;
       movf   RTCC,0     ; Controllo se finiti 20mS
       SKPZ              ;
       goto   DELA2M     ;
       movlw  128        ;
       movwf  RTCC       ;
       return            ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START  bsf    STAT,5     ; Seleziona SRAM banco 1
       movlw  b'0010'    ; RA0,RA2,RA3 out  RA1 in
       movwf  TR_A       ;
       movlw  b'00000000'; RB0...RB7 out
       movwf  TR_B       ;
       clrf   INTCON     ; Disabilita interrupt
       movlw  b'11000110'; Prescaler 1:128
       movwf   OPTIO     ; Copia W in OPTION
       bcf    STAT,5     ; Seleziona SRAM banco 0
       clrf   PORTA      ; Uscite della porta A tutte a 0
       clrf   PORTB      ; Uscite della porta B tutte a 0
;---------------------- main program ----------------------
       movlw  .128       ; Settaggio iniziale RTCC
       movwf  RTCC       ;
RIT1   movlw  .50        ; Carica 50 in W
       movwf  FR03       ; Copia W in FR03
RIT2   call   DELA2M     ; Vai alla subroutine DELA2M
       decfsz FR03       ; Decrementa FR03, salta se 0
       goto   RIT2       ; Vai a RIT2
       movlw  b'00000001'; Carica W
       xorwf  PORTB,1    ; Esegui l'OR-esclusivo tra W e la
                         ; porta B (toggla il led 0)
       goto   RIT1       ;
;-----------------------------------------------------------------
INTERP          ; Vettore interruzioni
       retfie           ;
;-----------------------------------------------------------------
       END
```

PROGRAMMA 5

```
TITLE 'PROG05: Acquisizione da pulsante per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H          ; Real Time Clock Counter
STAT    EQU 03H          ; Registro di stato
PORTA   EQU 05H          ; Porta A
PORTB   EQU 06H          ; Porta B
INTCON  EQU 0BH          ; Registro abilitazione interrupt
TR_A    EQU 85H          ; Tris A
TR_B    EQU 86H          ; Tris B
OPTIO   EQU 81H          ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH          ; Ritardo 20mS
FR02    EQU 0DH          ; Ritardo 20mS
FR03    EQU 0EH          ; Bit 0 = Memoria P2 per pressione
                         ; Bit 1 = Memoria P2 per rilascio
;-----------------------------------------------------------------
#define LED0   PORTB,0   ; Led 0 sulla porta B pin 0
#define P2     PORTA,1   ; Pulsante P2 su porta A pin 1
;-----------------------------------------------------------------
        ORG   0
        goto  START      ;Reset vector
        nop
        nop
        nop
        goto  INTERP     ;Interrupt vector
;---------------- subroutine RITARDO 20 mS ----------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------------
DELA2M bcf    FR03,0     ; Clear memoria P2 pressione
       bsf    FR03,1     ; Clear memoria P2 rilascio
WAIMS  clrwdt            ;
       btfsc  P2         ; Testo stato P2 per pressione
       bsf    FR03,0     ;
       btfss  P2         ; Testo stato P2 per rilascio
       bcf    FR03,1     ;
       movf   RTCC,0     ; Controllo se finiti 20mS
       SKPZ              ;
       goto   WAIMS      ;
       movlw  .128       ;
       movwf  RTCC       ;
       return            ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START  bsf    STAT,5     ; Seleziona SRAM banco 1
       movlw  b'0010'    ; RA0,RA2,RA3 out  RA1 in
       movwf  TR_A       ;
       movlw  b'00000000'; RB0...RB7 out
       movwf  TR_B       ;
       clrf   INTCON     ; Disabilita interrupt
       movlw  b'11000110'; Prescaler 1:128
       movwf  OPTIO      ; Copia W in OPTION
       bcf    STAT,5     ; Seleziona SRAM banco 0
       clrf   PORTA      ; Uscite della porta A tutte a 0
       clrf   PORTB      ; Uscite della porta B tutte a 0
       movlw  .128       ; Settaggio iniziale RTCC
       movwf  RTCC       ;
;---------------------- main program ----------------------
MAIN   call   DELA2M     ; Subroutine ritardo 20mS
       btfss  FR03,0     ; Testo P2 per pressione
       goto   ACCEND     ;
       btfsc  FR03,1     ; Testo P2 per rilascio
       goto   SPEGNI     ;
       goto   MAIN       ;
ACCEND bsf    LED0       ; Led 0 ON
       goto   MAIN       ;
SPEGNI bcf    LED0       ; Led 0 OFF
       goto   MAIN       ;
;-----------------------------------------------------------------
INTERP                   ; Vettore interruzioni
       retfie            ;
;-----------------------------------------------------------------
       END
```

PROGRAMMA 6

```
TITLE 'PROG06: Toggle con pulsante per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ; Ritardo 20mS
FR02    EQU 0DH         ; Ritardo 20mS
FR03    EQU 0EH         ; Bit 0 = Memoria P2 per pressione
                        ; Bit 1 = Memoria P2 per rilascio
                        ; Bit 2 = Memoria per toggle
;-----------------------------------------------------------------
#define LED0  PORTB,0   ; Led 0 sulla porta B pin 0
#define P2    PORTA,1   ; Pulsante P2 su porta A pin 1
;-----------------------------------------------------------------
        ORG   0
        goto  START     ;Reset vector
        nop
        nop
        nop
        goto  INTERP    ;Interrupt vector
; ---------- subroutine RITARDO 20 mS ----------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------------
DELA2M  bcf   FR03,0    ; Clear memoria P2 pressione
        bsf   FR03,1    ; Clear memoria P2 rilascio
WAIMS   clrwdt          ;
        btfsc P2        ; Testo stato P2 per pressione
        bsf   FR03,0    ;
        btfss P2        ; Testo stato P2 per rilascio
        bcf   FR03,1    ;
        movf  RTCC,0    ; Controllo se finiti 20mS
        SKPZ            ;
        goto  WAIMS     ;
        movlw .128      ;
        movwf RTCC      ;
        return          ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0010'   ; RA0,RA2,RA3 out  RA1 in
        movwf TR_A      ;
        movlw b'00000000' ; RB0...RB7 out
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        movlw b'11000110' ; Prescaler 1:128
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        clrf  PORTA     ; Uscite della porta A tutte a 0
        clrf  PORTB     ; Uscite della porta B tutte a 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf RTCC      ;
        bcf   FR03,2    ; Clear memoria toggle
;---------- main program ----------
MAIN    call  DELA2M    ; Subroutine ritardo 20mS
        btfss FR03,0    ; Testo P2 per pressione
        goto  ACCEND    ;
        btfsc FR03,1    ; Testo P2 per rilascio
        bcf   FR03,2    ; Clear memoria toggle
        goto  MAIN      ;
ACCEND  btfsc FR03,2    ; Vedo se memoria toggle ON
        goto  MAIN      ;
        bsf   FR03,2    ; Setto memoria toggle
        movlw b'00000001' ;
        xorwf PORTB,1   ; Toggle led 0
        goto  MAIN      ;
;-----------------------------------------------------------------
INTERP                  ; Vettore interruzioni
        retfie          ;
;-----------------------------------------------------------------
        END
```

In Figura 16 vediamo l'onda relativa alla pressione di un pulsante: la tensione assume subito il massimo valore (o il minimo, a seconda della presenza di resistenze di pull-up o di pull-down), ma dopo poco scende con andamento sinusoidale per poi risaliread una tensione inferiore della precedente e così via, fino all'annientarsi del fenomeno.

La durata di queste oscillazioni può variare da poche centinaia da microsecondi a diversi millisecondi. Durante prove di laboratorio, abbiamo constatato che un buon filtro software deve lavorare per almeno 20 mS per annientare queste oscillazioni.

Vediamo allora come implementare un programma che rileva la pressione di un pulsante senza prendere impulsi spuri e senza farsi influenzare dai rimbalzi.

Cerchiamo di capire la logica di questo programma. Il programma principale, chiama in continuazione la subroutine di attesa fine 20 mS, e non si preoccupa di testare direttamente lo stato del pulsante P2, ma va a vedere lo stato di due bit del registro FR03 che abbiamo impiegato come memorie per lo stato del pulsante stesso.

Quando richiamiamo la subroutine DELA2M, questi due bit vengono inizializzati rispettivamente a "0" ed a "1".

Durante il ciclo di attesa di fine tempo, il pulsante P2 viene testato in continuazione dalle due istruzioni BTFSC e BTFSS. Se il pulsante viene rilasciato anche per pochi microsecondi, la memoria relativa allo stato di premuto (FR03 bit 0) si cancella, quindi al ritorno della subroutine non accetteremo lo stato di pressione tasto.

Viceversa, se il pulsante viene premuto anche per pochi microsecondi, non rileveremo lo stato di rilascio pulsante.

Così facendo, abbiamo inserito un filtro software di durata precisa di 20 mS, filtro che, nel caso di implementazione hardware, avrebbe richiesto almeno un gruppo resistenza-condensatore e che non sarebbe stato così preciso.

A questo vantaggio, si somma la possibilità di implementare dei tempi di rilevamento pressione e rilascio completamente differenti e lunghi quanto si vuole.

In alcuni casi, per esempio, può essere utile non far partire un macchinario se prima non si è mantenuto premuto il pulsante di START per almeno N secondi. Situazioni di questo tipo sono facilmente gestibili via software, semplicemente inserendo dei registri che contano il numero di cicli da 20 mS fino ad arrivare al tempo desiderato.

Per esercizio provate a modificare il programma PROG05 per far in modo che il Led si accenda un secondo dopo la pressione del pulsante e si spenga 3 secondi dopo.

All'inizio, sembrerà una cosa complessa, ma con i programmi che avete già studiato, troverete il sistema per farlo.

Tabella 4. Capacità per oscillatori ceramici

Tipo di oscillatore	Resonator Frequency	Capacitor Range C1 = C2
XT	455 kHz 2,0 MHz 4,0 MHz	150÷330 pF 20÷330 pF 20÷330 pF
HS	8,0 MHz	20÷200 pF

Un toggle a pulsante

Vediamo ora come potremmo implementare un toggle a pulsante, ovvero come cambiare lo stato di una uscita (nel nostro caso sempre il Led 0) ad ogni pressione di un pulsante (P2).

Per far ciò, oltre ai problemi già trattati, si somma la memorizzazione dello stato di uscita.

Per implementare un flip-flop, abbiamo impiegato il bit numero 2 del registro di memoria FR03, lo stesso già utilizzato per la memoria di pressione e rilascio pulsante.

Il procedimento è banale: ad inizio programma si azzera la memoria del toggle e si entra nel ciclo principale.

Come avveniva precedentemente, si testa il pulsante P2 attraverso le memorie dei bit 0 e 1 del registro FR03.

Quando il pulsante non viene premuto, la memoria del toggle viene azzerata.

Quando, invece, il pulsante viene premuto, si va subito a testare lo stato della memoria di toggle: se questa è azzerata, vuol dire che il pulsante era stato rilasciato e che quindi si deve invertire (togglare) l'uscita del Led e, contemporaneamente, settare la memoria di toggle.

Se la memoria di toggle era già stata settata, l'inversione era già avvenuta e quindi si torna al ciclo principale senza effettuare altri cambiamenti. La memoria del toggle verrà poi azzerata con il primo rilascio del pulsante P2. Da notare che il pulsante P1 è quello che resetta fisicamente il microcontroller, quindi, premendolo, il programma si ritroverà ad eseguire nuovamente le istruzioni dalla label START, spegnendo tutti i Led.

Tabella 5. Capacità per oscillatori quarzati

Tipo di oscillatore	Frequenza	C1	C2
LP	32 kHz	30 pF	30÷50 pF
	100 kHz	15 pF	15 pF
	200 kHz	0÷15 pF	0÷15 pF
XT	100 kHz	15÷30 pF	200÷300 pF
	200 kHz	15÷30 pF	100÷200 pF
	455 kHz	15÷30 pF	15÷100 pF
	1 MHz	15÷30 pF	15÷30 pF
	2 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	10 MHz	15 pF	15 pF

Il reset hardware

Oltre ai vari reset di tipo software che possiamo avere (ad esempio la semplice istruzione "goto START"), quello più importante ai fini di un buon funzionamento è quello hardware, implementato sul piedino MCLR (Master CLeaR). Quando la tensione su questo pin scende ad uno "0" logico, il chip si pone in reset fino a quando sullo stesso pin non rileva un "1" logico.

La soluzione più semplice, e meno costosa, è quella applicata al circuito di Figura 15: un condensatore verso massa ed una resistenza sul positivo.

Quando viene fornita alimentazione al controller, sul pin MCLR avremo un basso livello fino a quando il condensatore non sarà completamente carico. Questa soluzione però crea inconvenienti quando si possono avere dei bruschi abbassamenti della tensione di alimentazione.

Allora, per prevenire questi casi, si debbono adottare soluzioni diverse, tra le quali quelle suggerite in Figura 17a e 17b. In entrambi i casi, in qualsiasi momento la tensione di alimentazione scenda sotto una soglia prefissata, il chip esegue un reset. La tensione di soglia viene impostata nel primo caso con un diodo zener, nel secondo caso con un partitore resistivo e con la soglia stessa sulla base del transistor Q1.

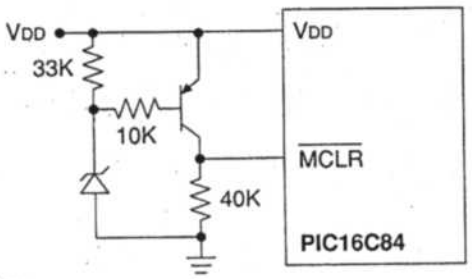
Il reset, per un microcontroller, è una fase essenziale per un buon funzionamento, poiché con esso vengono inizializzati correttamente tutti i registri dedicati e le porte vengono messe in three-state per tutta la durata del reset stesso.

Oltre al pin di reset, dobbiamo dedicare attenzione anche ai pin dell'oscillatore: se viene impiegato un gruppo RC, i valori consigliati per la resistenza vanno da 2.200 Ohm a 100 kΩ, mentre per il condensatore vanno da 10 pF a 1.000 pF.

È possibile adottare valori diversi, ma prove pratiche non lo consigliano.

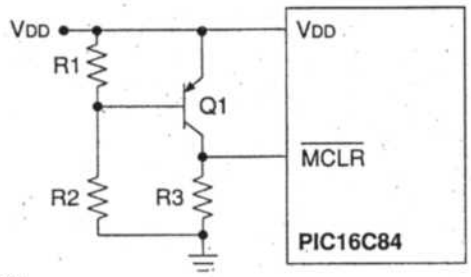
Nelle Tabelle 4 e 5 troviamo le capacità dei due condensatori da applicare sui pin dell'oscillatore rispettivamente nel caso di un oscillatore ceramico o quarzato.

continua



Notes:
1. This circuit will activate reset when VDD goes below (Vz + 0.7V) where Vz = Zener voltage.

Figura 17a. Reset con soglia a diodo zener



Notes:
1. This brown circuit is less expensive, albeit less accurate. Transistor Q1 turns off when VDD is below a certain level such that:

$$VDD \cdot \frac{R1}{R1 + R2} = 0.7\ V.$$

Figura 17b. Reset con soglia resistiva

LA PROGRAMMAZIONE IN ASSEMBLER

Il nostro corso di programmazione in assembler sta suscitando numerosi consensi in quanto aiuta gli appassionati del settore elettronico ad avvicinarsi al mondo dei microprocessori e della loro gestione

Andrea Sbrana - 5ª parte
su gentile concessione della Microlabs

Uno degli scogli maggiori per chi inizia a lavorare con i microcontroller, è l'interfacciamento con tastiere a matrice.

La cosa potrebbe a prima vista sembrare complessa, ma in realtà è implementabile con poche righe di software.

Se la tastiera fosse composta da n pulsanti con un capo in comune, o connesso al positivo oppure a massa, sapreste già come risolvere il problema, perché si tratterebbe soltanto di moltiplicare per n le routine di acquisizione da pulsante viste nella precedente puntata.

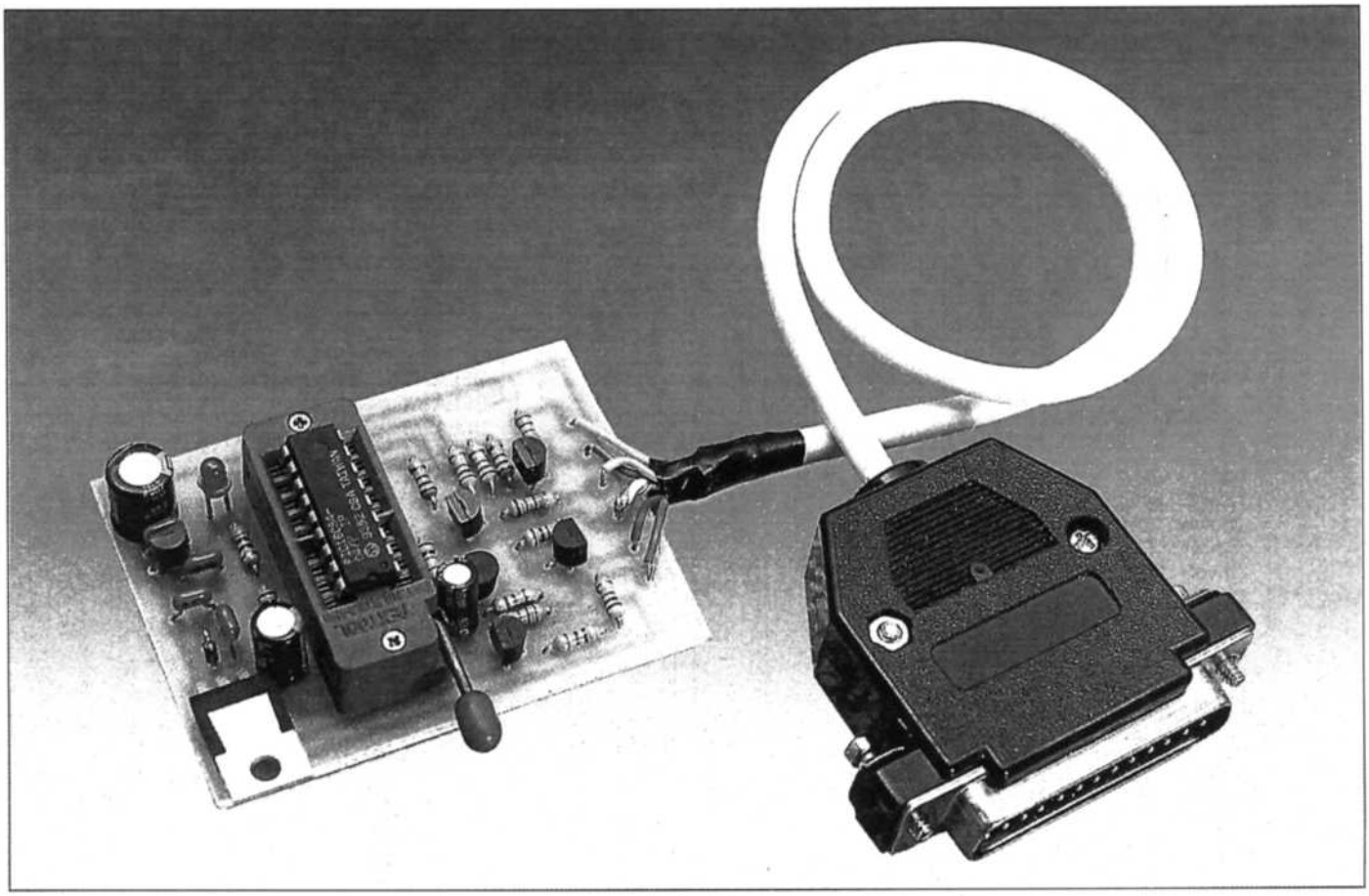
Con una tastiera a matrice, invece, non possiamo applicare le stesse routine, poiché avremmo conflittualità su pulsanti di una stessa riga oppure di una stessa colonna.

Cerchiamo allora di capire come è formata una tastiera a matrice, aiutandoci con lo schema di Figura 18.

Vediamo che per implementare una tastiera a matrice con 16 pulsanti, li dobbiamo connettere in modo tale da ottenere otto collegamenti: quattro per le righe della matrice e quattro per le colonne.

Nello schema vediamo anche resistenze di pull-up, ma queste sulle tastiere vere e proprie non ci sono.

In pratica, tali resistenze si trovano poi sul circuito elettrico, perché con esse è possibile realizzare un piccolo algoritmo in grado di decifrare se un tasto viene premuto e quale tasto sia.



PROGRAMMA 7

```
TITLE 'PROG07: Scansione tastiera per Progetto'
list F=INHX8M,P=16C84
;-----------------------------------------------------------
RTCC    EQU 01H             ; Real Time Clock Counter
STAT    EQU 03H             ; Registro di stato
PORTA   EQU 05H             ; Porta A
PORTB   EQU 06H             ; Porta B
INTCON  EQU 0BH             ; Registro abilitazione interrupt
TR_A    EQU 85H             ; Tris A
TR_B    EQU 86H             ; Tris B
OPTIO   EQU 81H             ; Registro OPTION
;-----------------------------------------------------------
FR01    EQU 0CH             ; Ritardo 20mS
FR02    EQU 0DH             ; Ritardo 20mS
FR03    EQU 0EH             ; Buffer pulsante premuto
;-----------------------------------------------------------
#define LED     PORTA,0     ; Led sulla porta A pin 0
#define C1      PORTB,4     ; Colonna 1
#define C2      PORTB,5     ; Colonna 2
#define C3      PORTB,6     ; Colonna 3
#define C4      PORTB,7     ; Colonna 4
;-----------------------------------------------------------
        ORG     0
        goto    START       ;Reset vector
        nop
        nop
        nop
        goto    INTERP      ;Interrupt vector
;-------------------- subroutine scansione tastiera --------
;Torna 0 se nessun tasto premuto, altrimenti il valore del tasto
;-----------------------------------------------------------
TASTI   movlw   b'11111110' ; Abilito scansione riga 1
        movwf   PORTB       ;
        clrwdt              ;
        btfss   C1          ; Test pulsante P1
        retlw   .1          ;
        btfss   C2          ; Test pulsante P2
        retlw   .2          ;
        btfss   C3          ; Test pulsante P3
        retlw   .3          ;
        btfss   C4          ; Test pulsante P4
        retlw   .4          ;
        movlw   b'11111101' ; Abilito scansione riga 2
        movwf   PORTB       ;
        clrwdt              ;
        btfss   C1          ; Test pulsante P5
        retlw   .5          ;
        btfss   C2          ; Test pulsante P6
        retlw   .6          ;
        btfss   C3          ; Test pulsante P7
        retlw   .7          ;
        btfss   C4          ; Test pulsante P8
        retlw   .8          ;
        movlw   b'11111011' ; Abilito scansione riga 3
        movwf   PORTB       ;
        clrwdt              ;
        btfss   C1          ; Test pulsante P9
        retlw   .9          ;
        btfss   C2          ; Test pulsante P10
        retlw   .10         ;
        btfss   C3          ; Test pulsante P11
        retlw   .11         ;
        btfss   C4          ; Test pulsante P12
        retlw   .12         ;
        movlw   b'11110111' ; Abilito scansione riga 4
        movwf   PORTB       ;
        clrwdt              ;
        btfss   C1          ; Test pulsante P13
        retlw   .13         ;
        btfss   C2          ; Test pulsante P14
        retlw   .14         ;
        btfss   C3          ; Test pulsante P15
        retlw   .15         ;
        btfss   C4          ; Test pulsante P16
        retlw   .16         ;
        movlw   b'11111111' ; Disabilito scansione tastiera
        movwf   PORTB       ;
        retlw   .0          ; Torna dopo la CALL
;-------------------- subroutine RITARDO 20 mS -------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------
DELA2M  clrwdt              ;
        movf    RTCC,0      ; Controllo se finiti 20mS
        SKPZ                ;
        goto    DELA2M      ;
        movlw   .128        ;
        movwf   RTCC        ;
        return              ; Ritorna dopo la CALL
;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START   bsf     STAT,5      ; Seleziona SRAM banco 1
        movlw   b'0000'     ; RA0,RA1,RA2,RA3 out
        movwf   TR_A        ;
        movlw   b'11110000' ; RB0...RB3 out  RB4...RB7 in
        movwf   TR_B        ;
        clrf    INTCON      ; Disabilita interrupt
        movlw   b'01000110' ; Prescaler 1:128
        movwf   OPTIO       ; Copia W in OPTION
        bcf     STAT,5      ; Seleziona SRAM  banco 0
        movlw   .128        ; Settaggio iniziale RTCC
        movwf   RTCC        ;
        movlw   b'0001'     ; Led off
        movwf   PORTA       ;
        movlw   b'11111111' ; Disabilita scansione
        movwf   PORTB       ;
;-------------------- main program -------------------------
MAIN    call    DELA2M      ; Subroutine ritardo 20mS
        call    TASTI       ; Scansiono tastiera
        movwf   FR03        ; Buffer pulsante premuto
        xorlw   .0          ;
        btfsc   STAT,2      ;
        goto    MAIN        ;
        bcf     LED         ; Accensione led
LPSIG   call    DELA2M      ; Ritardo complessivo x 100mS
        call    DELA2M      ;
        call    DELA2M      ;
        call    DELA2M      ;
        call    DELA2M      ;
        decfsz  FR03        ; Testo se scaduto numero cicli
        goto    LPSIG       ;
        bsf     LED         ; Spegnimento led
WAIFIN  call    DELA2M      ;
        call    TASTI       ;
        xorlw   .0          ; Verifica tasto ancora premuto
        btfss   STAT,2      ;
        goto    WAIFIN      ;
        goto    MAIN        ;
;-----------------------------------------------------------
INTERP                      ; Vettore interruzioni
        retfie              ;
;-----------------------------------------------------------
    END
```

Il metodo di scansione

Per ottenere il numero del tasto premuto, ci avvaliamo di software che dobbiamo implementare nel microcontroller e che, ripetutamente, va a testare le otto linee della tastiera in un certo modo.

Diciamo subito che, per ottenere ciò, le otto linee devono essere divise in due gruppi da quattro: nel nostro esempio le colonne saranno viste dal controller come ingressi (ecco il perché delle resistenze di pull-up), mentre le righe saranno predisposte come uscite.

In Figura 19 troviamo il diagramma temporale di come, generalmente, vengono abilitate le uscite relative alle righe della matrice: quattro abilitazioni formano, un ciclo detto "ciclo di scansione".

Alla partenza, tutte le uscite del controller sono ad alto livello, quindi sui quattro ingressi troveremo quattro livelli logici alti. Il ciclo inizia con il portare a basso livello l'uscita relativa alla riga 1: allora, se andassimo a leggere ora gli ingressi, avremmo uno "0" nel caso del pulsante premuto, un "1" se tutto rimane come prima.

Ad ogni ingresso colonna, corrisponderà il relativi pulsante abilitato dalla riga 1. Eseguita la scansione della riga 1, tale linea viene riportata ad alto livel-

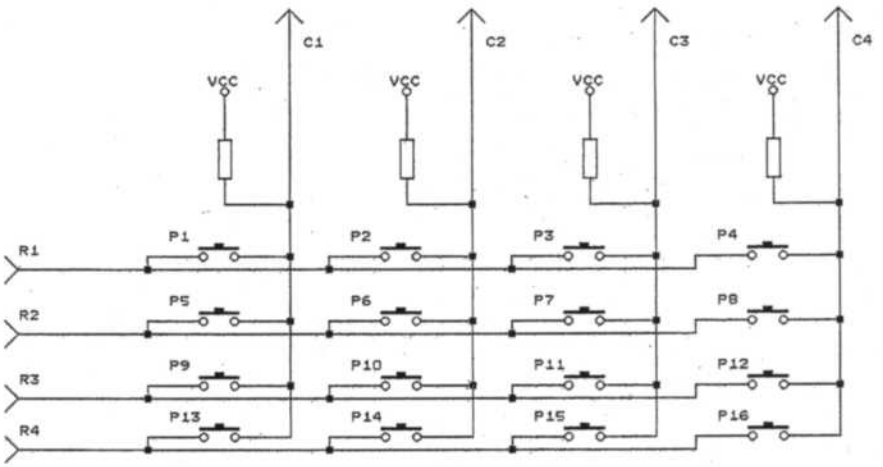


Figura 18. Schema di una tastiera a matrice di 4x4 tasti

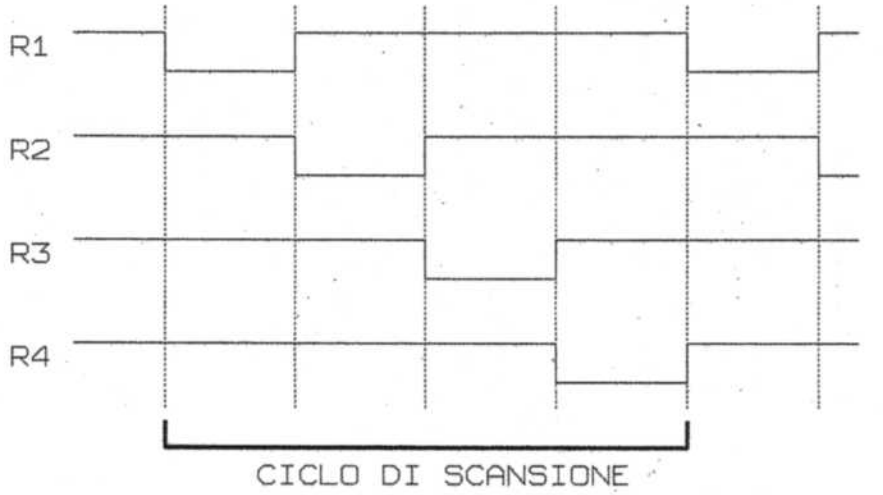


Figura 19. Diagramma temporale del ciclo di scansione

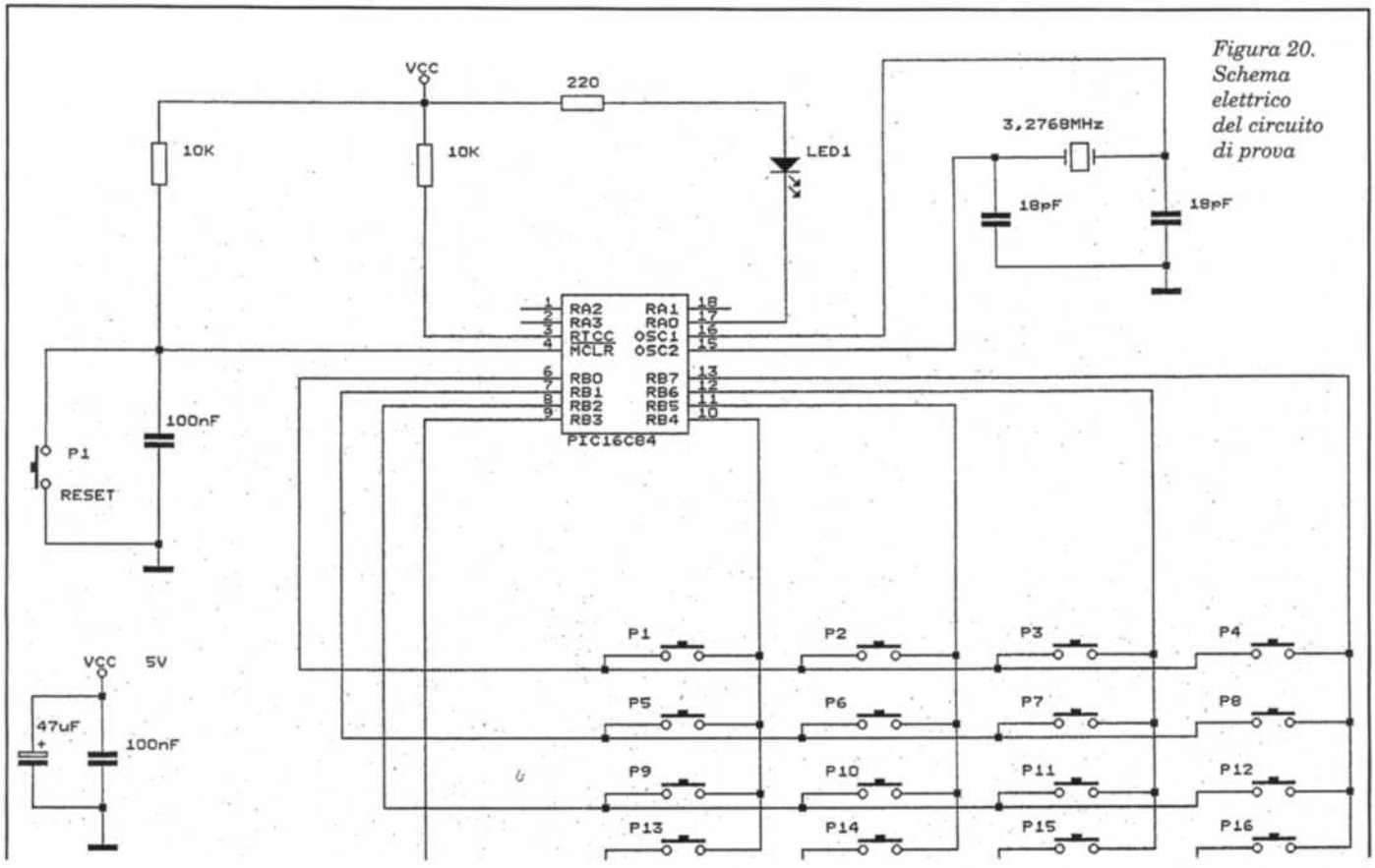


Figura 20. Schema elettrico del circuito di prova

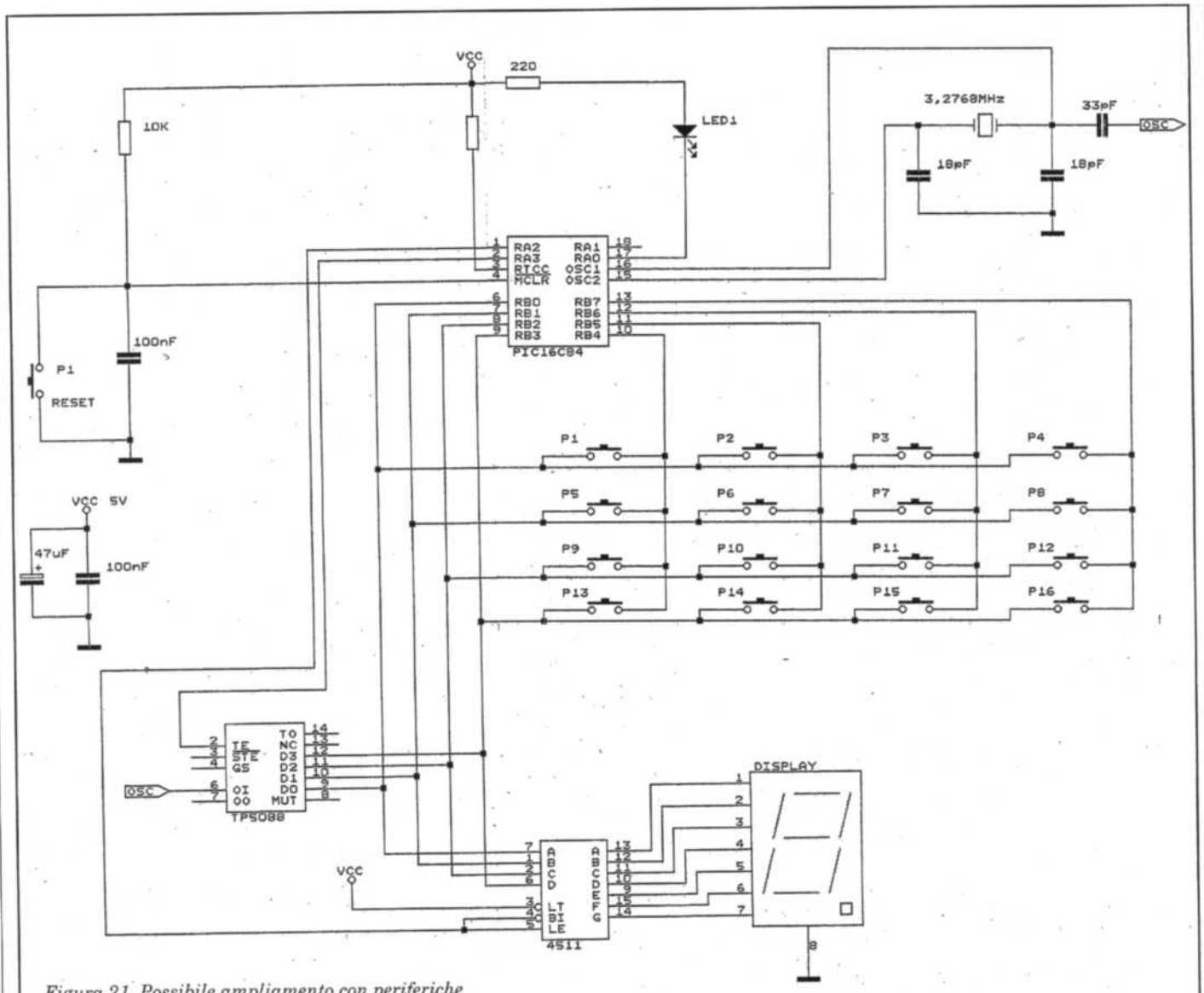


Figura 21. Possibile ampliamento con periferiche

lo e si passa alla riga 2. Il procedimento è identico al precedente, solo che questa volta, saranno abilitati soltanto i pulsanti relativi a quella riga. Il ciclo termina quando anche la quarta riga è stata abilitata.

I tempi di scansione possono essere anche molto rapidi, ovviamente anche in funzione della distanza della tastiera dal micro e dal valore delle resistenze di pull-up.

Lo stesso meccanismo avrebbe funzionato anche invertendo le righe con le colonne, oppure inserendo delle resistenze di pull-down e abilitando i pulsanti con segnali ad alto livello.

Questo sistema ci consente, come vedremo avanti, di sfruttare sia gli ingressi che le uscite in comune con altre periferiche, che potranno essere abilitate separatamente con pin di selezione.

Il circuito di verifica

In Figura 20 troviamo lo schema elettrico del circuito che dovremo realizzare per verificare il funzionamento del programma PROG07.ASM.

Con questo programma, vedremo come implementare la subroutine di scansione della tastiera.

Esaminiamo il programma nelle sue funzioni: il Led resta spento fino a quando non viene premuto un pulsante della tastiera. Quando ciò avviene, il Led si accende per un tempo pari a 100 mS moltiplicato il numero del pulsante premuto; dopodiché il Led si spegne fino a quando non viene premuto un altro pulsante. Se il pulsante viene lasciato premuto, il Led si accenderà solo dopo il suo rilascio e la pressione di un nuovo pulsante.

La prima modifica ai precedenti programmi sta nella direzione delle porte: adesso la porta A è completamente di uscita, mentre la porta B è per metà di ingresso e per metà di uscita.

Il registro OPTION viene caricato con il bit 7 a "0", perché vogliamo che siano abilitate le resistenze di pull-up sui pin di ingresso della porta B. Così facendo ci siamo risparmiati quattro resistenze da montare sul circuito.

Il ciclo principale del programma, prevede la chiamata alla subroutine di attesa 20 mS e poi alla nuova subroutine "TASTI". Questa subroutine, torna con il valore 0 in W se nessun tasto è stato premuto, mentre, se premiamo un pulsante, torna caricando in W il valore corrispondente al tasto premuto.

Per esempio il pulsante P1 ritornerà il valore 1, il pulsante P8 il valore 8 ecc.

Il valore che la subroutine pone in W, viene copiato nel registro FR03 e viene confrontato con zero. Se è uguale, nessun tasto è stato premuto e quindi si torna al MAIN, viceversa si accende il Led e si attende un tempo pari a 100 mS per il numero copiato. Al termine il Led viene spento e si ripete la chiamata a "TASTI" per attendere il rilascio del pulsante.

La procedura di attesa rilascio è complementare a quella di attesa pressione, ma implementata con la stessa tecnica dello XOR.

Proviamo allora a capire come lavorare con l'operazione XORLW (oppure XORWF). Come ben sappiamo, l'operazione di or esclusivo, restituisce valore "0" se i due (o più) ingressi sono uguali, altrimenti restituisce valore "1".

Quindi, facendo lo XOR tra il valore decimale 0 ed il numero copiato in FR03, il bit numero 2 (zero bit) del registro di stato (STAT) varrà 1 se i due numeri sono uguali, 0 altrimenti.

Con l'istruzione BTFSC STAT,2 si testa questo bit e il gioco è fatto.

Ma passiamo ora allo studio della subroutine "TASTI". Inizialmente si pongono le quattro uscite della porta B (relative alle quattro righe della matrice) tutte ad alto livello. Poi si pone a basso livello la RB0 (riga 1) e si vanno a testare in sequenza le quattro colonne corrispondenti ai quattro pulsanti della prima riga.

Se una di queste vale zero, il corrispondente pulsante è stato premuto e quindi si troverà l'istruzione RETLW N, dove N è il numero di ritorno in W.

Se, invece, nessuno di questi pulsanti viene premuto, si riporta l'uscita della riga 1 ad alto livello e si procede nello stesso modo con la riga 2, fino ad arrivare alla quarta riga.

La subroutine "TASTI" è universale, nel senso che è possibile sia ridurla ad un minor numero di righe e/o colonne, sia aumentarla, compatibilmente con il numero di pin disponibili.

Con questo software avrete la possibilità di generare degli impulsi da 100 mS a 1,6 secondi molto precisi, semplicemente impostandone il numero sulla tastiera.

In Figura 21, troviamo un suggerimento per poter impiegare le stesse uscite delle righe della tastiera con altre due periferiche che vengono abilitate dalle uscite RA2 e RA3 del micro.

In comune abbiamo i quattro segnali D0, D1, D2, D3.

Il TP5088 è un generatore di toni DTMF, mentre il 4511 è un pilota per display a sette segmenti con ingresso binario.

Per esercizio provate a far accendere sul display i numeri da 0 a 8 in funzione del tasto premuto.

ATTENZIONE: anche se la cosa risulta banale, in realtà non lo è, poiché dovrete confrontarvi con il rilascio del pulsante stesso!

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Il nostro corso è arrivato al suo giro di boa: chi ci ha seguito è ormai diventato autonomo nella gestione e nella realizzazione di piccoli programmi. In questa sesta parte, approfondiamo gli aspetti legati alla gestione di display

Andrea Sbrana - 6ª parte
su gentile concessione della Microlabs

In questa sezione del corso, impareremo a pilotare display a sette segmenti. Come tutti sappiamo, la soluzione di inserire un display a sette segmenti per una qualsiasi indicazione, è, fra le tante possibili, una delle meno dispendiose e di sicuro effetto.

Quando, alcuni anni fa, non si impiegavano i microcontroller per le piccole e medie applicazioni, per gestire i display erano necessari integrati dedicati, come quello visto lo scorso mese, in grado di convertire un numero binario in codifica a sette segmenti. In alcuni casi, venivano programmate delle PAL o delle GAL, per ottenere anche un minimo di ottimizzazione delle porte logiche. Con l'avvento dei microcontroller, gestire un display a sette segmenti è diventato semplicissimo, e, come vedremo, anche più di uno.

La soluzione più rapida

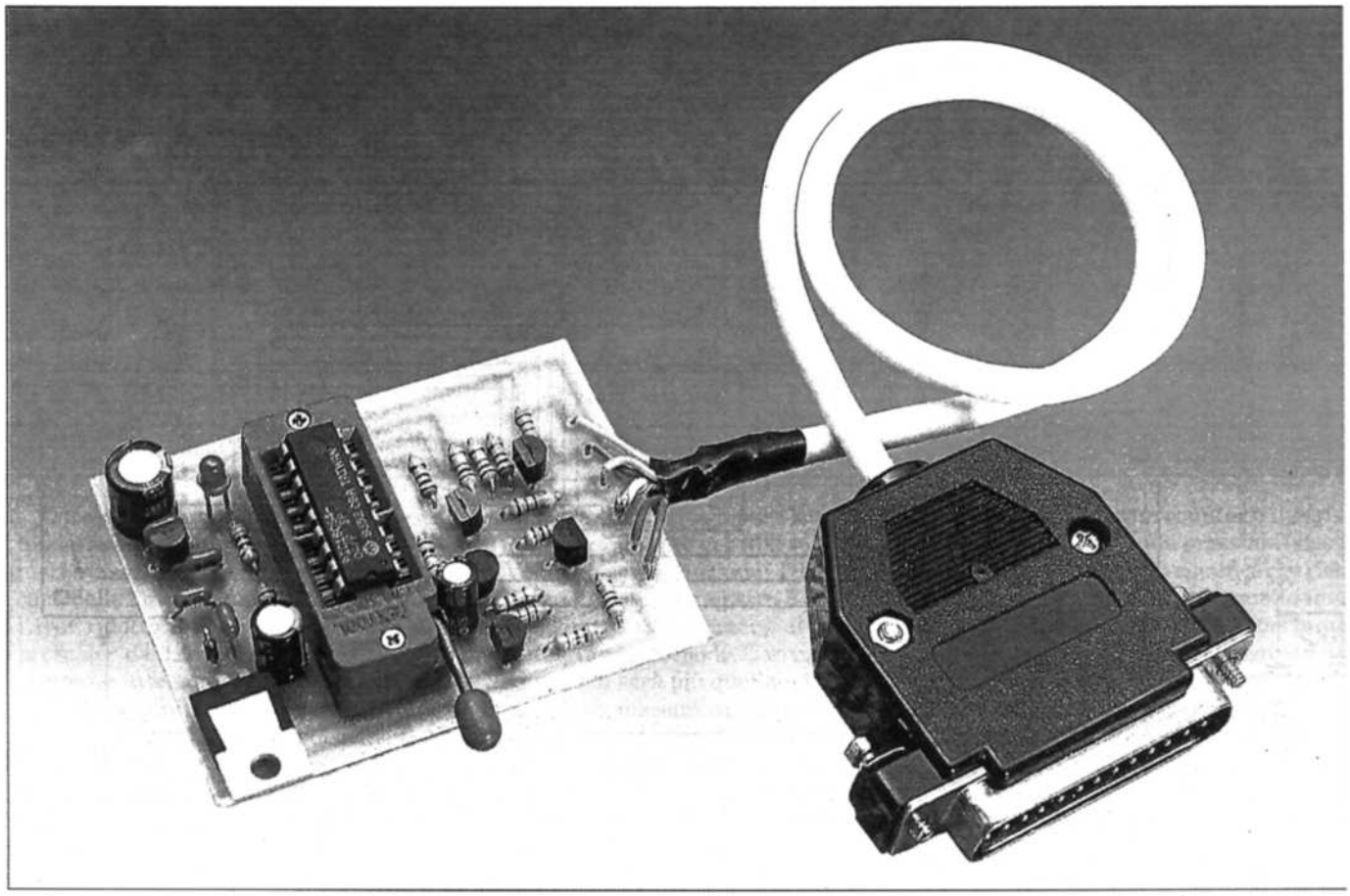
In Figura 22, troviamo lo schema ele trico necessario per poter eseguire esp rimenti con un display. Da notare che numeri sui pin del display sono in succe sione come le corrispondenti lettere d segmenti, ovvero il numero 1 corrispo de alla lettera "A", il numero 2 alla "B" così via fino alla "F". Il punto decima non è stato utilizzato.

Il display ha il positivo in comun mentre i sette segmenti si accendo portandoli verso la massa.

Le resistenze da 220 Ω riducono l'a sorbimento di ogni segmento.

Il PIC quindi, per accendere uno d segmenti, dovrà settare a 0 il pin cor spondente.

Il programma PROG08 visualizza numeri da 0 a 9, in crescendo, con passi un secondo.



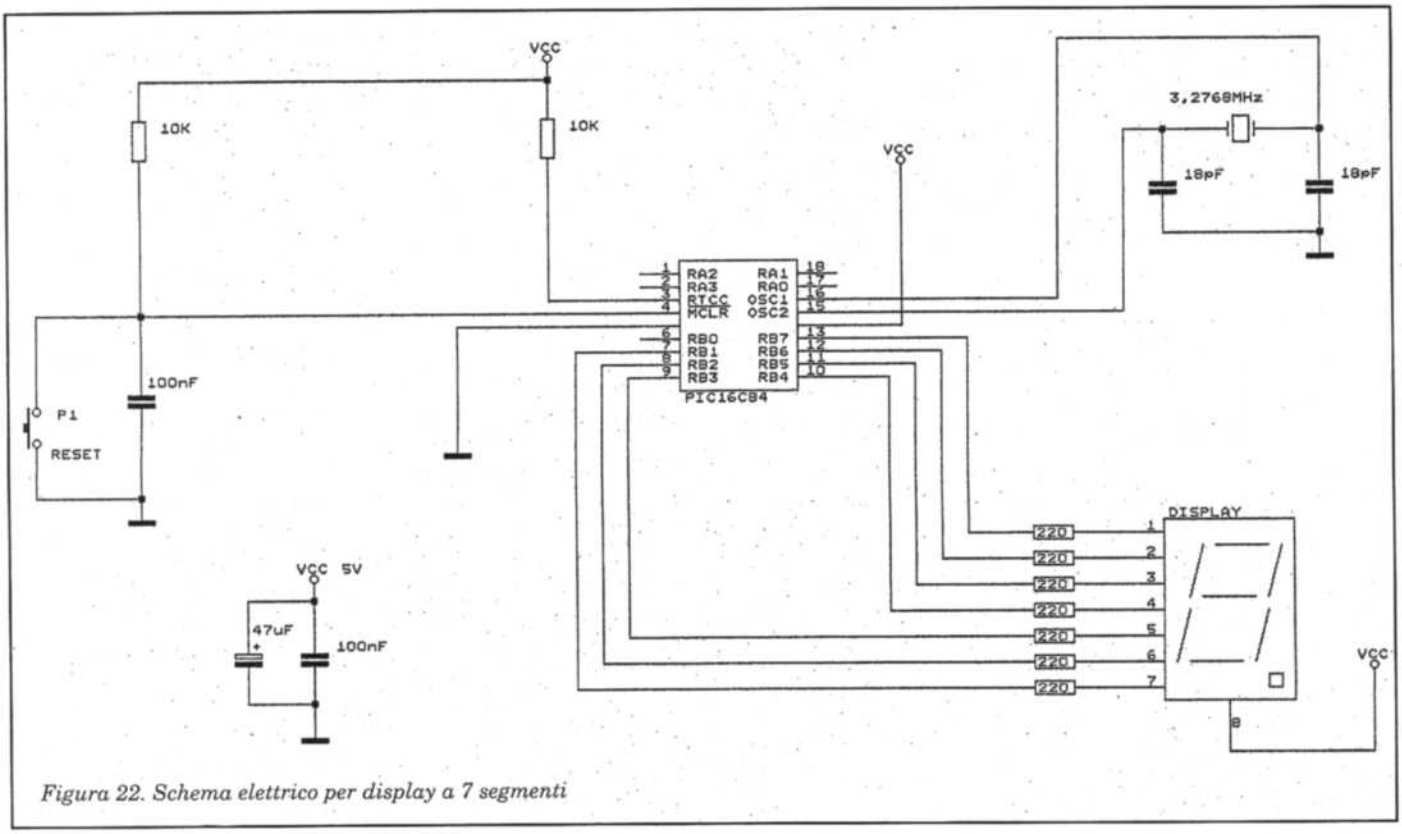


Figura 22. Schema elettrico per display a 7 segmenti

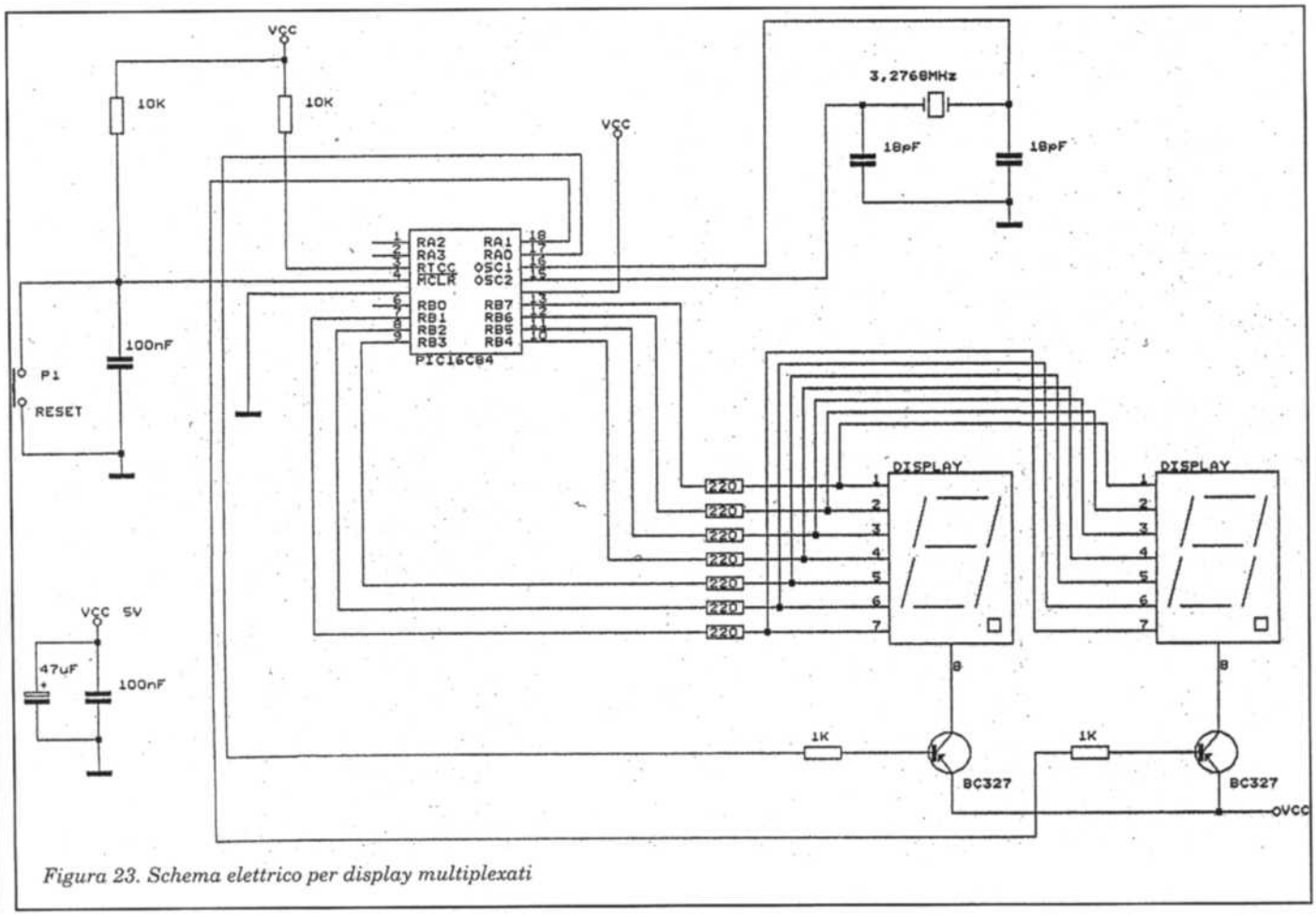


Figura 23. Schema elettrico per display multiplexati

PROGRAMMA 8

```
TITLE 'PROG08: Controllo display 7 seg. per Progetto'
 list F=INHX8M,P=16C84
;-------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-------------------------------------------------------------
FR01    EQU 0CH         ; Ritardo 1S
FR02    EQU 0DH         ; Numero da visualizzare
;-------------------------------------------------------------
;---------- Definizione costanti per display ----------
ZER_O   EQU 003H        ; Codifica numero 0: 00000011
UNO     EQU 09FH        ; Codifica numero 1: 10011111
DUE     EQU 025H        ; Codifica numero 2: 00100101
TRE     EQU 00DH        ; Codifica numero 3: 00001101
QUATT   EQU 099H        ; Codifica numero 4: 10011001
CINQUE  EQU 049H        ; Codifica numero 5: 01001001
SEI     EQU 041H        ; Codifica numero 6: 01000001
SETTE   EQU 01FH        ; Codifica numero 7: 00011111
OTTO    EQU 001H        ; Codifica numero 8: 00000001
NOVE    EQU 009H        ; Codifica numero 9: 00001001
;-------------------------------------------------------------
        ORG   0
        goto  START     ;Reset vector
        nop
        nop
        nop
        goto  INTERP    ;Interrupt vector

;---------- subroutine RITARDO 5 mS ----------
; con frequenza 3,2768 MHz
;-------------------------------------------------------------
DELA5M  clrwdt          ;
        movf  RTCC,0    ; Controllo se finiti 5mS
        SKPZ            ;
        goto  DELA5M    ;
        movlw .128      ;
        movwf RTCC      ;
        return          ; Ritorna dopo la CALL
;---------- subroutine presentazione numero su display ----------
DISPLA  movf  FR02,0    ; Copio FR02 in w
        addwf PCL,1     ; Sommo w al program counter
        retlw ZER_O     ; Visualizza 0
        retlw UNO       ; Visualizza 1
        retlw DUE       ; Visualizza 2
        retlw TRE       ; Visualizza 3
        retlw QUATT     ; Visualizza 4
        retlw CINQUE    ; Visualizza 5
        retlw SEI       ; Visualizza 6
        retlw SETTE     ; Visualizza 7
        retlw OTTO      ; Visualizza 8
        retlw NOVE      ; Visualizza 9
;-------------------------------------------------------------
; START PROGRAM
;-------------------------------------------------------------
START   bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0000'   ; RA out
        movwf TR_A      ;
        movlw b'00000000' ; RB out
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        movlw b'11000100' ; Prescaler 1:32
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf RTCC      ;
        movlw b'0000'   ; Porta A off
        movwf PORTA     ;
        movlw b'11111111' ; Disabilita segmenti display
        movwf PORTB     ;
;---------- main program ----------
MAIN    clrf  FR02      ; Numero iniziale
MAIN1   call  DISPLA    ; Visualizza numero
        movwf PORTB     ; su porta B
        movlw .200      ; Conteggi da 5mS per 1S
        movwf FR01      ;
LOOP    call  DELA5M    ; Subroutine ritardo 5mS
        decfsz FR01     ; Testo se scaduto 1S
        goto  LOOP      ;
        incf  FR02      ; Incrmento numero da visualizzare
        movlw .10       ; Copio 10 in W
        xorwf FR02,0    ; Testo se FR02 = 10
        btfss STAT,2    ;
        goto  MAIN1     ; <> 10
        goto  MAIN      ; = 10
;-------------------------------------------------------------
INTERP                  ; Vettore interruzioni
        retfie          ;
;-------------------------------------------------------------
    END
```

Dato che il programma ha le stesse inizializzazioni dei precedenti, ci soffermeremo soltanto sulle righe che introducono delle novità. La prima consiste nell'aver ridotto il fattore di divisione del prescaler da 128 a 32. In questo modo, otteremo attese di 5mS precisi. In più sono state eliminate le resistenze di pull-up sulla porta B, visto che questa serve solo da output.

Il numero da visualizzare viene mantenuto in FR02 ed inizialmente vale zero.

Nel ciclo MAIN1, si richiama la subroutine DISPLA, che torna la codifica da copiare sulla porta B per visualizzare il numero contenuto in FR02 sul display.

Vediamo come funziona: DISPLA copia il contenuto del registro FR02 in W, poi aggiunge tale numero al program counter. In questo modo, l'istruzione successiva, non sarà più quella immediatamente dopo, ma sarà calcolata sommando il numero da visualizzare al program counter. Vediamo un esempio pratico supponendo che il numero contenuto nel registro FR02 sia "3". Allora, in W viene copiato un 3. L'istruzione ADDWF PCL,1 somma al program counter un 3.

Poiché il program counter è il registro che punta sempre alla prossima istruzione da eseguire, questo significa che l'istruzione successiva non sarà quella immediatamente dopo la ADDWF, ma la quarta (la prima se avessimo sommato 0).

Questa istruzione fa uscire dalla subroutine caricando il valore corrispondente alla label TRE nel registro W.

Al ritorno dalla subroutine, l'istruzione successiva è la MOVWF PORTB, che copierà il valore contenuto in w sulla porta B, facendo accendere i led corrispondenti al numero 3.

PROGRAMMA 9

```
TITLE 'PROG09: Multiplexer display 7 seg. per Progetto'
 list F=INHX8M,P=16C84
;-------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-------------------------------------------------------------
FR01    EQU 0CH         ; Ritardo 1S
FR02    EQU 0DH         ; Numero da visualizzare sx
FR03    EQU 0EH         ; Numero da visualizzare dx
;-------------------------------------------------------------
#define DISP1 PORTA,0   ; Abilitazione display 1
#define DISP2 PORTA,1   ; Abilitazione display 2
;---------- Definizione costanti per display ----------
ZER_O   EQU 003H        ; Codifica numero 0: 00000011
UNO     EQU 09FH        ; Codifica numero 1: 10011111
DUE     EQU 025H        ; Codifica numero 2: 00100101
TRE     EQU 00DH        ; Codifica numero 3: 00001101
QUATT   EQU 099H        ; Codifica numero 4: 10011001
CINQUE  EQU 049H        ; Codifica numero 5: 01001001
SEI     EQU 041H        ; Codifica numero 6: 01000001
SETTE   EQU 01FH        ; Codifica numero 7: 00011111
OTTO    EQU 001H        ; Codifica numero 8: 00000001
NOVE    EQU 009H        ; Codifica numero 9: 00001001
;-------------------------------------------------------------
        ORG   0
        goto  START     ;Reset vector
        nop
        nop
        nop
        goto  INTERP    ;Interrupt vector

;---------- subroutine RITARDO 5 mS ----------
; con frequenza 3,2768 MHz
;-------------------------------------------------------------
DELA5M  clrwdt          ;
        movf  RTCC,0    ; Controllo se finiti 5mS
        SKPZ            ;
        goto  DELA5M    ;
        movlw .128      ;
        movwf RTCC      ;
        return          ; Ritorna dopo la CALL
;---------- subroutine presentazione numero su display ----------
DISPLA  addwf PCL,1     ; Sommo w al program counter
        retlw ZER_O     ; Visualizza 0
        retlw UNO       ; Visualizza 1
        retlw DUE       ; Visualizza 2
        retlw TRE       ; Visualizza 3
        retlw QUATT     ; Visualizza 4
        retlw CINQUE    ; Visualizza 5
        retlw SEI       ; Visualizza 6
        retlw SETTE     ; Visualizza 7
        retlw OTTO      ; Visualizza 8
        retlw NOVE      ; Visualizza 9
;---------- subroutine gestione multiplexer ----------
VISUAL  movf  FR02,0    ; Numero sx
        call  DISPLA    ; Numero sx su porta B
        movwf PORTB     ;
        bcf   DISP1     ; Abilita display 1
        call  DELA5M    ; Attesa 5mS
        bsf   DISP1     ; Disabilita display 1
        movf  FR03,0    ; Numero dx
        call  DISPLA    ; Numero dx su porta B
        movwf PORTB     ;
        bcf   DISP2     ; Abilita display 2
        call  DELA5M    ; Attesa 5mS
        bsf   DISP2     ; Disabilita display 2
        movlw b'11111111' ; Spegni segmenti
        movwf PORTB     ;
        return          ;
;-------------------------------------------------------------
; START PROGRAM
;-------------------------------------------------------------
START   bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0000'   ; RA out
        movwf TR_A      ;
        movlw b'00000000' ; RB out
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        movlw b'11000100' ; Prescaler 1:32
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf  RTCC     ;
        movlw b'1111'   ; Display off
        movwf PORTA     ;
        movlw b'11111111' ; Disabilita segmenti display
        movwf PORTB     ;
;---------- main program ----------
MAIN    clrf  FR02      ; Numero iniziale sx
        movlw .9        ; Numero iniziale dx
        movwf FR03      ;
MAIN1   movlw .100      ; Conteggi da 10mS per 1S
        movwf FR01      ;
LOOP    call  VISUAL    ; Subroutine visualizzaz. 10mS
        decfsz FR01     ; Testo se scaduto 1S
        goto  LOOP      ;
        incf  FR02      ; Incrmento numero sx
        decf  FR03      ; Decremento numero dx
        movlw .10       ; Copio 10 in W
        xorwf FR02,0    ; Testo se FR02 = 10
        btfss STAT,2    ;
        goto  MAIN1     ; <> 10
        goto  MAIN      ; = 10
;-------------------------------------------------------------
INTERP                  ; Vettore interruzioni
        retfie          ;
;-------------------------------------------------------------
        END
```

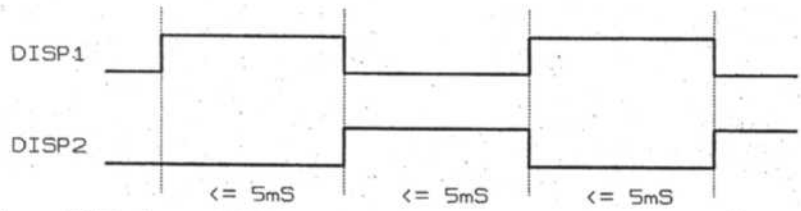


Figura 24. Timing per l'abilitazione dei due display

Le label da ZER_O a NOVE infatti, decodificano il numero che rappresentano nei sette segmenti del display.

La label UNO, ad esempio, pone a zero il bit 5 e il bit 6, corrispondenti proprio ai due segmenti B e C relativi al numero "1". In questo modo, possiamo modificare il valore sul display in qualsiasi momento, ad esempio facendo corrispondere allo zero la lettera "A", all'uno la lettera "b" e così via, compatibilmente con la disponibilità del display. Quello che abbiamo visto è il metodo più semplice ed efficace per implementare tabelle con i controller della Microchip. Ma vediamo un'altra serie di istruzioni che confrontano due valori: abbiamo necessità di sapere quando il numero dei conteggi è giunto a 9 e successivamente a 10, poiché in questo caso dobbiamo resettare il contatore ripartendo da zero. Dopo aver incrementato il contatore, copiamo il numero 10 nel registro W, poi ne facciamo lo XOR con il registro FR02, stando attenti a collocare il risultato in W (perché se lo mettessimo in FR02, il contatore verrebbe corrotto perdendo il suo valore originale).

Quello che ci interessa è andare a vedere se il risultato è uguale o diverso da 0. Nel primo caso, il registro FR02 è caricato con il valore 10, nel secondo con un numero diverso da 10. Quindi, nel primo caso si resetta il contatore, nel secondo si prosegue con la visualizzazione del numero e successiva attesa di un secondo.

La gestione di un multiplexer

Abbiamo già visto come vengono gestite le tastiere, ora vedremo che una cosa analoga deve essere implementata per la gestione di più display in multiplexer.

Che cosa significa multiplexer?

In questo caso, significa che dobbiamo controllare più display (nel nostro esempio 2) con le linee dei sette segmenti in comune e comandi di abilitazione separati.

In Figura 23 troviamo lo schema relativo alla nostra soluzione. Come si vede chiaramente, i display hanno i segmenti in comune, mentre il comune non è più connesso direttamente al positivo, ma ognuno ha una propria linea di abilitazione, collegata alla porta A tramite transistor PNP che servono ad erogare la corrente necessaria a tutti e sette i segmenti di ciascun display. In Figura 24 vediamo, invece, come avviene l'abilitazione dei due display: entrambi vengono attivati alternativamente per un tempo inferiore o uguale a 5 mS.

Ovviamente, quando viene abilitato il display numero 1, sulla porta B dovrà essere presente la decodifica corrispondente a quel display e così pure per il display numero 2. Se, ad esempio, volessimo visualizzare il numero "23", dovremmo prima copiare sulla porta B la decodifica relativa alla cifra "2", poi abilitare il display numero uno per 5 mS, disabilitarlo, portare sulla porta B la decodifica relativa alla cifra "3", abilitare il display 2 per 5 mS, disabilitarlo e tornare all'inizio del ciclo.

Ogni volta che viene richiamata la VISAUL, quindi, il tempo che trascorre equivale a 10mS, perché 5 mS servono al display uno e altri 5 mS servono al display numero due. Per ottenere allora il tempo di scorrimento di un secondo, si dovranno attendere 100 chiamate alla VISUAL (100 x 10 mS = 1.000 mS = 1S).

Sotto vediamo il programma PROG09 che implementa il multiplexer richiesto, visualizzando sul display di sinistra i numeri da 0 a 9 in crescendo e, su quello di destra, i numeri da 9 a 0 in decrescendo. Anche in questo programma, abbiamo utilizzato l'operazione di XOR per testare la fine del ciclo di visualizzazione.

La subroutine VISUAL è quella che, di fatto, fa accendere i due display nei tempi e nei modi precedentemente definiti.

Dobbiamo stare attenti, però, al numero di chiamate annidate che inseriamo, poichè il PIC16C84 ne consente otto, ma la famiglia PIC16C5X, ne consente solamente due, quindi fare attenzione a convertire un programma da PIC16C84 a PIC16C5X.

In sole 15 istruzioni, la VISUAL riesce a pilotare i due display. Se i display fossero stati 4 o 6 (6 solamente su PIC a 28 pin minimo) il numero di istruzioni sarebbe aumentato proporzionalmente di 15 e di 30 unità. Come vedete, quindi, il classico display a 4 cifre per timer ed orologi può essere implementato con sole 30 righe di software.

Ovviamente, se poi volete inserire anche i puntini lampeggianti, le cose si complicano un minimo (si dovrà utilizzare il pin RB0) e si dovrà tener conto di quale display è interessato al lampeggio e della frequenza di lampeggio, ma ormai dovreste essere già in grado di eseguire da voi, senza il nostro aiuto, le modifiche necessarie.

In Figura 25 suggeriamo la tempistica necessaria ad un multiplexing con quattro display, ricordando che 5 mS è il massimo tempo di attesa possibile, prima che i display appaiano lampeggianti per i tempi di persistenza di un immagine sulla retina.

Da prove effettuate, ma che potrete eseguire anche voi, abbiamo ottenuto i migliori risultati con tempistiche prossime al millisecondo.

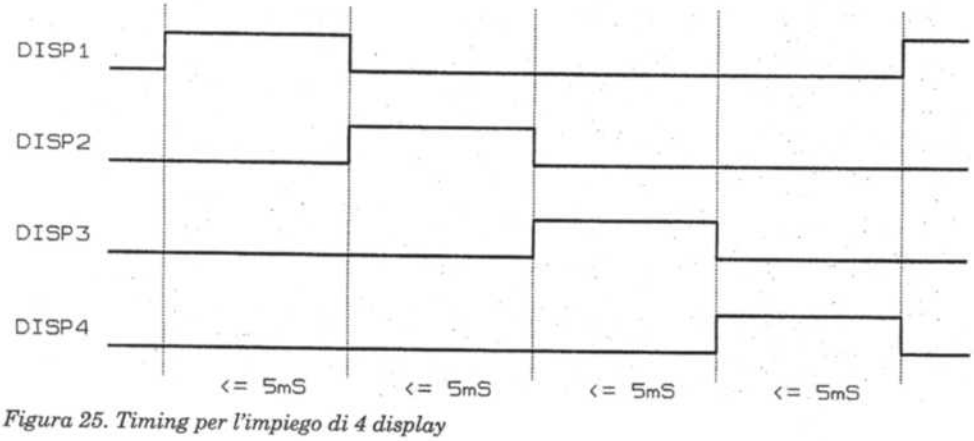


Figura 25. Timing per l'impiego di 4 display

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Proseguiamo come di consueto il nostro corso dedicato alla programmazione in assembler dei PIC, i microprocessori che stanno ottenendo un ottimo consenso da parte degli appassionati di elettronica. Ricordiamo che in occasione della prima puntata abbiamo presentato lo schema di un semplice programmatore

Paola Sbrana - 7ª parte
su gentile concessione della Microlabs

Come già anticipato nelle prime puntate, il PIC16C84 non solo ha la memoria di programma in tecnologia EEPROM, ma possiede anche 64 registri da 8 bit di identica struttura.

Questo vuol dire che, oltre alla riprogrammabilità del controller senza necessità di lampade a ultravioletti, è possibile immagazzinarvi dei dati che non saranno persi anche dopo aver tolto l'alimentazione.

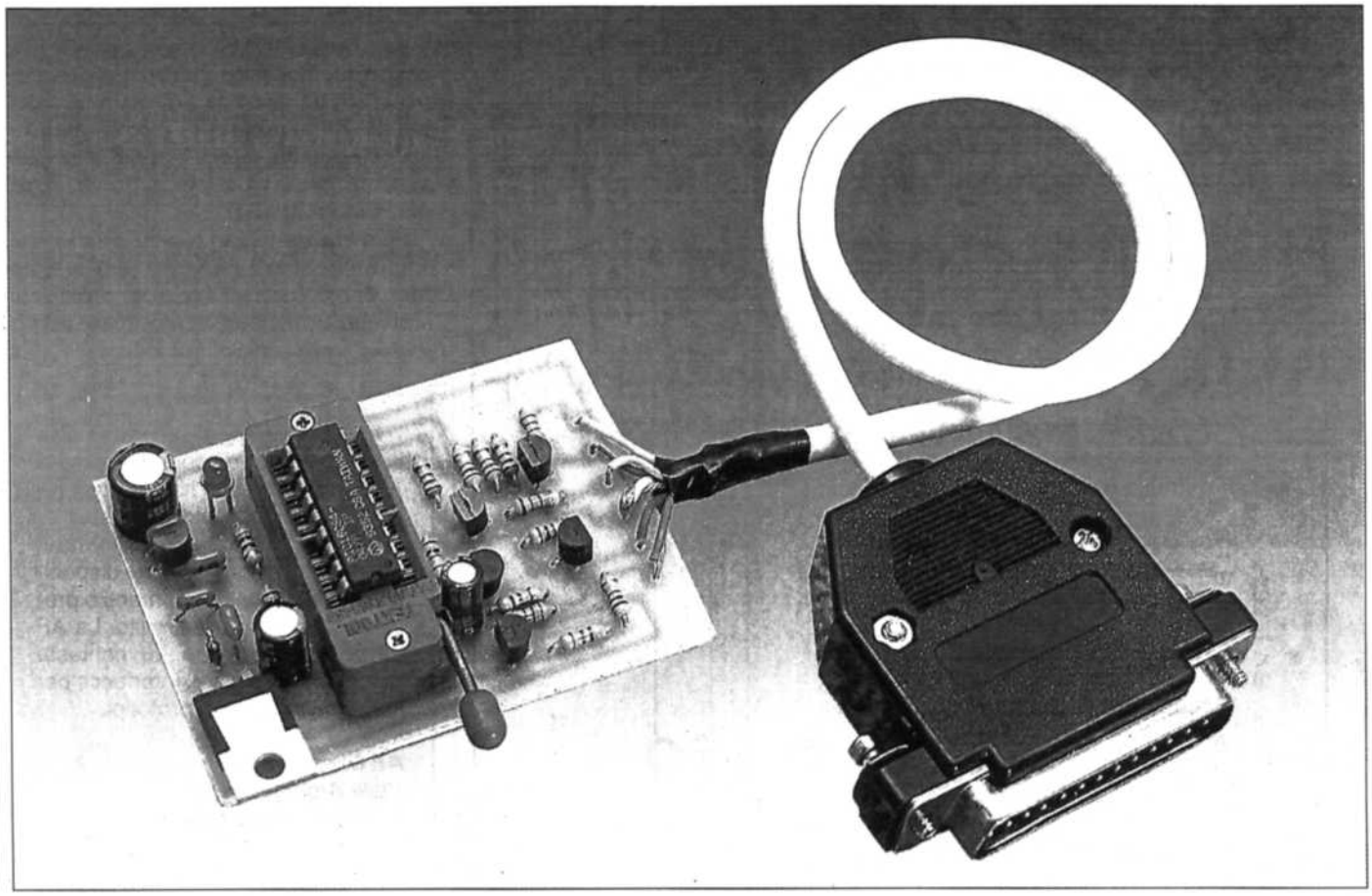
Questa proprietà, fa preferire il 16C84 ad altri in varie applicazioni, prima tra tutte quella dei telecomandi ad autoapprendimento o a roling-code (codice variabile).

Tutti gli altri controller che non hanno questa possibilità, devono essere affiancati da una EEPROM seriale esterna, con conseguente spreco di spazio, sia fisico, che di linee di programma, poiché l'interfacciamento con le EEPROM seriali avviene con protocolli da implementare via software.

Con il PIC16C84, invece, la scrittura o la lettura di una cella di memoria (byte) avviene semplicemente con pochissime istruzioni dedicate.

L'hardware dedicato

Il PIC16C84 ha quattro registri dedicati che gestiscono l'intera memoria EEPROM e le operazioni su di essa: EECON1, EECON2, EEDATA e EEADR.



Il registro EEDATA è il buffer di scambio per la lettura e la scrittura della memoria: per scrivere un byte in una cella, il byte stesso viene prima copiato in EEDATA. Viceversa, per leggere il contenuto di una cella, dopo aver dato alcune informazioni che vedremo, troviamo il dato nel registro EEDATA. Il numero di celle (byte) gestite è di 64 ed il loro indirizzamento va da 00h a 3Fh. Quando un byte viene scritto (o riscritto), prima viene automaticamente cancellato dal micro stesso, per garantire una perfetta memorizzazione del dato. Il tempo di memorizzazione è di circa 10 mS ed è controllato da un timer dedicato interno al controller. Il tempo dato varia comunque con la temperatura e la tensione di alimentazione. Un'importante opzione, consente di proteggere non solo la memoria di programma da letture non autorizzate, ma anche la memoria dei dati.

Il registro EEADR è il registro dedicato dove inserire l'indirizzo della cella su cui eseguire l'operazione e, ovviamente può indirizzare fino a 256 byte di memoria, ma soltanto i primi 64 sono disponibili. In Figura 26 troviamo la segmentazione del registro EECON1: il bit 0 (RD) dà il via alla lettura di una cella. Il tempo necessario è quello di una istruzione.

Per questa operazione, il bit deve essere settato via software, poi, al termine, viene azzerato dall'hardware del micro.

Il bit numero 1 (WR) invece, dà il via al ciclo di scrittura di una cella. Anche in questo caso, viene settato via software e azzerato dall'hardware a fine operazione. Con il bit numero 2 (WREN) è possibile proteggere i dati della EEPROM da accidentali scritture, perché se posto a "0" inibisce l'operazione di scrittura. Se settato, la consente.

Il bit numero 3 (WRERR) dà indicazione dell'avvenuta memorizzazione o meno.

Se azzerato, l'operazione si è conclusa correttamente, se settato, la scrittura non è avvenuta, ad esempio perché sopraggiunto un MCLR oppure per intervento del watchdog. Il bit numero 4 (EEIF) è simile al precedente, ma con lo stato invertito e deve essere azzerato via software. Gli ultimi tre bit non vengono usati. Il registro EECON2 invece non è un registro fisico vero e proprio, ma soltanto una mappatura per il chip.

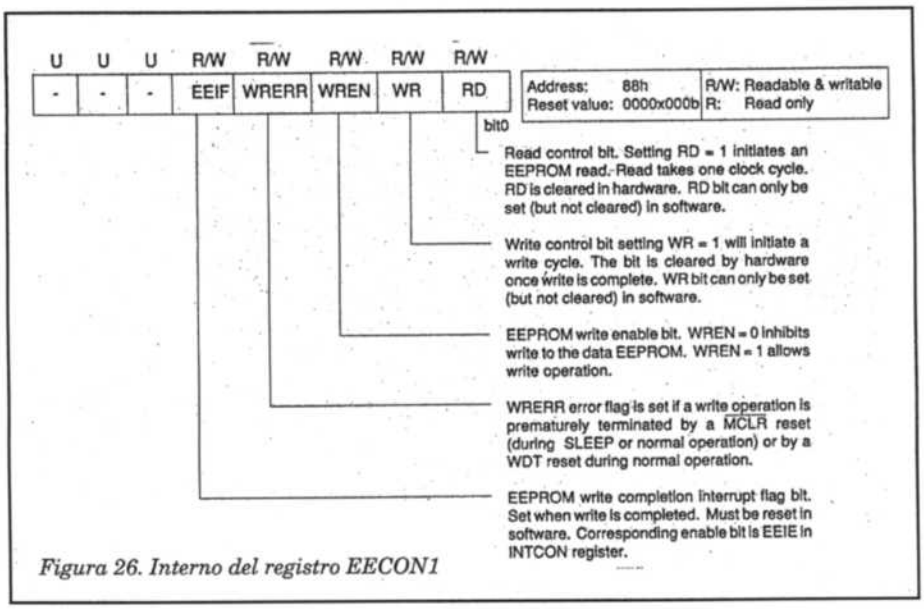


Figura 26. Interno del registro EECON1

Il nostro programma di test

In Figura 27 vediamo il circuito elettrico da realizzare per fare alcuni esperimenti con la memoria EEPROM interna.

Sono stati connessi quattro pulsanti sugli ingressi RB4÷RB7 e quattro Led sulle uscite RB0÷RB3, mentre il clock è sempre a 3,2768 MHz. I quattro pulsanti sono stati connessi su quei pin per un ben preciso motivo: quando spiegheremo le varie sorgenti di interrupt, una di queste arriverà dal cambio di stato di un pin della porta B<4÷7>. Ma passiamo al programma PROG10: nelle definizioni sono stati inseriti i nomi dei registri dedicati e dei quattro pulsanti.

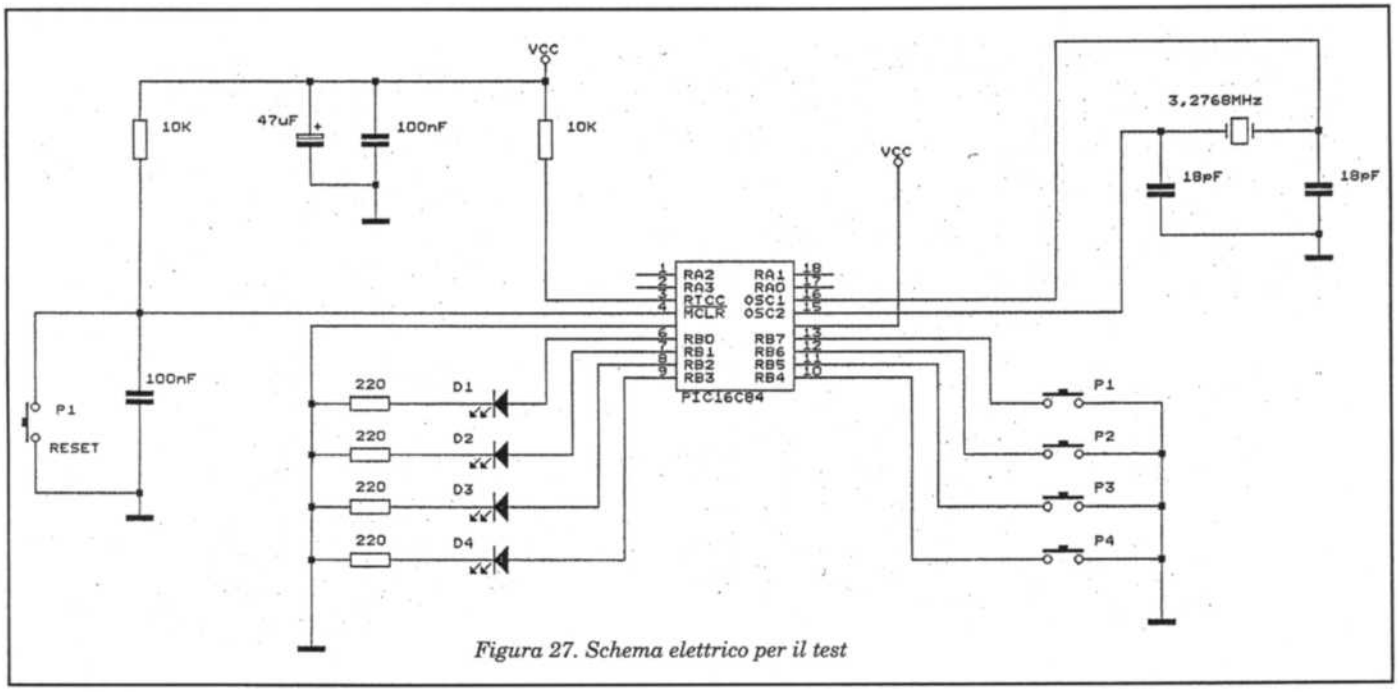


Figura 27. Schema elettrico per il test

PROGRAMMA 10

```
TITLE 'PROG10: Prova EEPROM interna per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ; Buffer per pressione tasti
FR02    EQU 0DH         ;
FR03    EQU 0EH         ;
;-----------------------------------------------------------------
#define P1 PORTB,7      ; Pulsante 1
#define P2 PORTB,6      ; Pulsante 2
#define P3 PORTB,5      ; Pulsante 3
#define P4 PORTB,4      ; Pulsante 4
;-----------------------------------------------------------------
        ORG   0
        goto  START     ; Reset vector
        nop
        nop
        nop
        goto  INTERP    ; Interrupt vector

;-------------- subroutine lettura EEPROM ----------------------
LEGGI   clrf  EADR      ;
        bsf   STAT,5    ; Page 1
        bsf   ECON1,0   ; Leggi EEPROM byte indirizzo 00h
        bcf   STAT,5    ; Page 0
        nop             ;
        movf  EDATA,0   ; Carica w con il dato letto
        movwf PORTB     ; Copia stato led
        return          ;
;-------- subroutine scrittura in EEPROM ---------------------
MEMO    clrf  EADR      ; Memorizza all'indirizzo 0
        movf  PORTB,0   ; Copia la porta B in W
        movwf EDATA     ; Copia W in EDATA
        bsf   STAT,5    ; Page 1
        bsf   ECON1,2   ; WREN Abilita scrittura
        movlw 55H       ; Carica 55h in w
        movwf ECON2     ; Copia W in ECON2
        movlw 0AAH      ; Carica aah in W
        movwf ECON2     ; Copia W in ECON1
        bsf   ECON1,1   ; Abilita scrittura
LPW1    clrwdt          ; Attendi fine scrittura
        btfss ECON1,4   ;
        goto  LPW1      ;
        bcf   ECON1,4   ; Clear fine scrittura
        bcf   STAT,5    ; Pagina 0
RIL     call  DELA2M    ; Attesa rilascio pulsante
        btfss P1        ;
        goto  RIL       ;
        btfss P2        ;
        goto  RIL       ;
        btfss P3        ;
        goto  RIL       ;
        btfss P4        ;
        goto  RIL       ;
        goto  MAIN      ;

;-------------- subroutine RITARDO 20 mS ----------------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------------
DELA2M  bcf   FR01,7    ; Memoria P1
        bcf   FR01,6    ; Memoria P2
        bcf   FR01,5    ; Memoria P3
        bcf   FR01,4    ; Memoria P4
LPDEL   clrwdt          ;
        btfsc P1        ; Testo P1
        bsf   FR01,7    ; Cancello memoria
        btfsc P2        ; Testo P2
        bsf   FR01,6    ; Cancello memoria
        btfsc P3        ; Testo P3
        bsf   FR01,5    ; Cancello memoria
        btfsc P4        ; Testo P4
        bsf   FR01,4    ; Cancello memoria
        movf  RTCC,0    ; Controllo se finiti 20mS
        SKPZ            ;
        goto  LPDEL     ;
        movlw .128      ;
        movwf RTCC      ;
        return          ; Ritorna dopo la CALL

;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0000'   ; RA out
        movwf TR_A      ;
        movlw b'11110000' ; RB0..RB3 out  RB4..RB7 in
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        movlw b'01000110' ; Prescaler 1:128
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf RTCC      ;
        movlw b'0000'   ;
        movwf PORTA     ;
        call  LEGGI     ; Leggi ultimo stato memorizzato
;-------------------- main program ---------------------------
MAIN    call  DELA2M    ; Attesa 20mS + test pulsanti
        btfss FR01,7    ; Testo se premuto P1
        goto  PULS1     ;
        btfss FR01,6    ; Testo se premuto P2
        goto  PULS2     ;
        btfss FR01,5    ; Testo se premuto P3
        goto  PULS3     ;
        btfss FR01,4    ; Testo se premuto P4
        goto  PULS4     ;
        goto  MAIN      ;
PULS1   movlw b'11110001' ; Accensione led 1
        movwf PORTB     ;
        goto  MEMO      ; Memorizza nuovo stato
PULS2   movlw b'11110010' ; Accensione led 2
        movwf PORTB     ;
        goto  MEMO      ; Memorizza nuovo stato
PULS3   movlw b'11110100' ; Accensione led 3
        movwf PORTB     ;
        goto  MEMO      ; Memorizza nuovo stato
PULS4   movlw b'11111000' ; Accensione led 4
        movwf PORTB     ;
        goto  MEMO      ; Memorizza nuovo stato
;-----------------------------------------------------------------
INTERP                  ; Vettore interruzioni
        retfie          ;
;-----------------------------------------------------------------
        END
```

Il software è stato implementato in modo che, premendo un pulsante, si accenda il Led corrispondente e si spengano gli altri, ed in più venga memorizzato il nuovo stato delle uscite nella prima cella della EEPROM dati.

Se viene tolta e ridata alimentazione, il controller va a leggersi l'ultimo stato scritto nella EEPROM e riconfigura adeguatamente le uscite.

Analizziamo ora le due routine di lettura e scrittura memoria, dato che l'interfacciamento con i pulsanti è già stato visto nelle precedenti applicazioni.

La subroutine LEGGI inizia con la selezione del secondo banco di registri, dove si trova il registro EECON1 ed azzera il bit 0 di quest'ultimo per avviare la fase di lettura.

Dobbiamo premettere che in precedenza, nel registro EEADR deve essere stato caricato l'indirizzo della cella da leggere (nel nostro caso lo 00h).

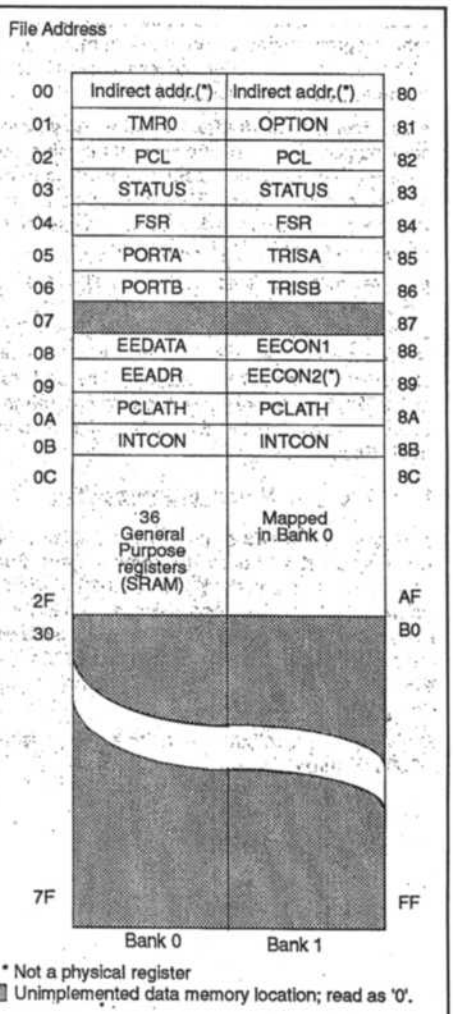


Figura 28. Mappa dei due banchi di registri

Tabella 6. Indirizzo dei registri e loro stato dopo un reset

Register	Address	Power-on Reset	MCLR Reset during: – normal operation – SLEEP WDT timeout during normal operation	Wake up from SLEEP: – through interrupt – through WDT timeout
W	–	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000h	0000h	PC + 1[2]
STATUS	03h	0001 1xxx	000? ?uuu[3]	uuu? ?uuu[3]
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	05h	---u uuuu	---u uuuu	---u uuuu
PORTB	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu[1]
INDF	80h	---- ----	---- ----	---- ----
OPTION	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000h	0000h	PC + 1
STATUS	83h	0001 1xxx	000? ?uuu[3]	uuu? ?uuu[3]
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	----1 1111	----u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 0000	---0 ?000	---0 ?uuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu[1]

Legend: u = unchanged x = unknown - = unimplemented bit, read as '0', ? = value depends on condition

Note 1: One or more bits in INTCON will be affected (to cause wake-up)

2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h)

3: Table 8-5 lists the reset value for each specific condition

In Figura 28 trovate l'indirizzo di tutti i registri del banco 0 e del banco 1. Poi si ripassa al primo banco di registri e si attende un ciclo. L'istruzione successiva, copia il valore trovato in EEDATA nel registro w (tale valore è proprio lo stato letto nella prima locazione di memoria).

Con l'ultima istruzione si copia lo stato da w alla porta B, poi si torna al programma principale.

Con queste poche istruzioni, abbiamo letto la cella di memoria di indirizzo 00h e ne abbiamo copiato il contenuto sulla porta B (chiaramente solo per quei pin settati come uscite).

La subroutine MEMO è invece leggermente più complessa, poiché sono necessari alcuni passi di programmazione senza i quali la scrittura non avviene correttamente.

Con la prima istruzione, carichiamo il registro EEADR con l'indirizzo della cella in cui scrivere, ovvero l'indirizzo 00h.

Con le due successive invece copiamo lo stato della porta B nel registro EEDATA. Ci sono poi una serie di istruzioni che servono a far partire il ciclo di scrittura e che terminano con il settaggio del bit 1 del registro EECON1.

Da qui si entra in un loop di attesa in cui si testa continuamente il bit 4 sempre di EECON1. Quando questo si setterà, allora il ciclo di scrittura sarà terminato e dovremo azzerare tale bit via software.

Le istruzioni seguenti, invece, non fanno altro che attendere che tutti e quattro i pulsanti siano stati rilasciati.

Questo semplice programma, pur nella sua limitatezza applicativa, pone in risalto la metodologia per lavorare con la memoria EEPROM del PIC16C84.

Nella Tabella 6, possiamo vedere l'indirizzamento fisico dei vari registri ed in più il loro stato dopo un reset.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Entriamo ancora di più nel vivo del nostro corso di programmazione in assembler dei PIC, i processori della famiglia Risc che stanno ottenendo, anche grazie alle numerose pubblicazioni su Progetto Elektor, un ottimo successo di pubblico

Paola Sbrana - 8ª parte
su gentile concessione della Microlabs

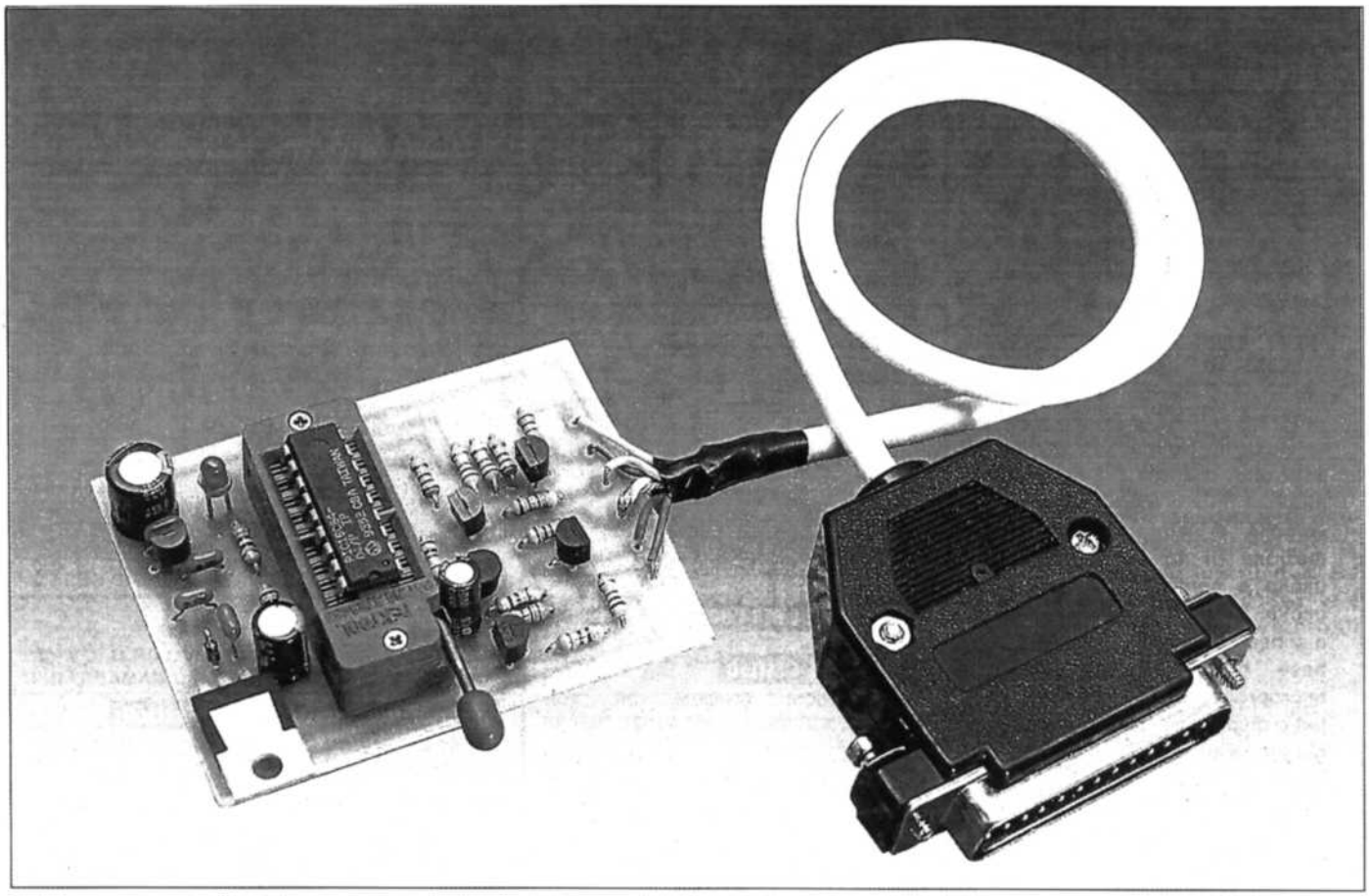
In questa nuova puntata, impareremo a far generare al PIC dei segnali di bassa frequenza, prima semplici, poi più complessi. Precisiamo subito che, a meno di non disporre di un convertitore digitale/analogico, le forme d'onda prelevabili su un pin del controller non potranno che essere di tipo quadrato, poi facilmente modellabili con appositi filtri bassa-basso. Molti lettori ci hanno scritto che era loro intenzione progettare un antifurto auto, con sirena incorporata, altri che volevano realizzare uno sweep, altri ancora che avevano necessità di implementare un tasto morse con parole già memorizzate.

Chiaramente, non è possibile accontentare ogni singola richiesta, ma vogliamo ugualmente permettere a queste persone di realizzare autonomamente il software che serve loro, senza essere obbligati quindi ad acquistare dei chip già programmati, ovviamente più costosi del chip vergine.

Aprendo adesso una doverosa parentesi, aggiungiamo che vi accorgerete da soli del perché un chip programmato costi abbastanza caro: sommate le ore che vi serviranno per realizzare il vostro prototipo e constaterete che risulteranno diversi giorni di lavoro.

Lo schema elettrico

Passiamo, quindi, ad analizzare in Figura 29 lo schema elettrico necessario alla prova dei software che andremo a



PROGRAMMA 11

```
TITLE 'PROG11: Tono a 1KHz per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------
RTCC    EQU 01H          ; Real Time Clock Counter
PCL     EQU 02H          ; Program Counter
STAT    EQU 03H          ; Registro di stato
PORTA   EQU 05H          ; Porta A
PORTB   EQU 06H          ; Porta B
EDATA   EQU 08H          ; Data EEPROM
EADR    EQU 09H          ; Address EEPROM
ECON1   EQU 88H          ; Stato delle operaz. in EEPROM
ECON2   EQU 89H          ; Vedi ECON1
INTCON  EQU 0BH          ; Registro abilitazione interrupt
TR_A    EQU 85H          ; Tris A
TR_B    EQU 86H          ; Tris B
OPTIO   EQU 81H          ; Registro OPTION
;-----------------------------------------------------------
FR01    EQU 0CH          ;
FR02    EQU 0DH          ;
FR03    EQU 0EH          ;
;-----------------------------------------------------------
#define P1 PORTB,7       ; Pulsante 1
#define TONO PORTA,2     ; Uscita segnale BF
;-----------------------------------------------------------
        ORG   0
        goto  START      ;Reset vector
        nop
        nop
        nop
        goto  INTERP     ;Interrupt vector

;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START   bsf   STAT,5     ; Seleziona SRAM banco 1
        movlw b'0000'    ; RA out
        movwf TR_A       ;
        movlw b'11110000'; RB0..RB3 out  RB4..RB7 in
        movwf TR_B       ;
        clrf  INTCON     ; Disabilita interrupt
        movlw b'01000110'; Enable pull-up su porta B
        movwf OPTIO      ; Copia W in OPTION
        bcf   STAT,5     ; Seleziona SRAM banco 0
        movlw .128       ; Settaggio iniziale RTCC
        movwf RTCC       ;
        clrf  PORTA      ; Poni a zero le uscite su A
        clrf  PORTB      ; Poni a zero le uscite su B
;------------------------ main program ---------------------
MAIN    clrwdt ; Azzero watchdog
        bcf   TONO       ; Tono off
        btfsc P1         ; Testo se P1 premuto
        goto  MAIN       ; Pulsante non premuto
                         ;
LPTONO  movlw b'0100'    ; Togglo tono
        xorwf PORTA,1    ; su porta A
        movlw .100       ; Carico 100 in W
        movwf FR01       ; Carico W in FR01
LP1     clrwdt           ; Azzero watchdog
        nop              ; Nessuna operazione
        decfsz FR01      ; Decrem. FR01 e testo se zero
        goto  LP1        ; Non zero
        btfsc P1         ; Testo se P1 ancora
        goto  MAIN       ; Pulsante non premuto
        goto  LPTONO     ;
;-----------------------------------------------------------
INTERP                   ; Vettore interruzioni
        retfie           ;
;-----------------------------------------------------------
        END
```

PROGRAMMA 12

```
TITLE 'PROG12: Tono con duty cycle variabile per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------
RTCC    EQU 01H          ; Real Time Clock Counter
PCL     EQU 02H          ; Program Counter
STAT    EQU 03H          ; Registro di stato
PORTA   EQU 05H          ; Porta A
PORTB   EQU 06H          ; Porta B
EDATA   EQU 08H          ; Data EEPROM
EADR    EQU 09H          ; Address EEPROM
ECON1   EQU 88H          ; Stato delle operaz. in EEPROM
ECON2   EQU 89H          ; Vedi ECON1
INTCON  EQU 0BH          ; Registro abilitazione interrupt
TR_A    EQU 85H          ; Tris A
TR_B    EQU 86H          ; Tris B
OPTIO   EQU 81H          ; Registro OPTION
;-----------------------------------------------------------
FR01    EQU 0CH          ;
FR02    EQU 0DH          ;
FR03    EQU 0EH          ;
;-----------------------------------------------------------
#define P1 PORTB,7       ; Pulsante 1
#define TONO PORTA,2     ; Uscita segnale BF
;-----------------------------------------------------------
        ORG   0
        goto  START      ;Reset vector
        nop
        nop
        nop
        goto  INTERP     ;Interrupt vector

;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START   bsf   STAT,5     ; Seleziona SRAM banco 1
        movlw b'0000'    ; RA out
        movwf TR_A       ;
        movlw b'11110000'; RB0..RB3 out  RB4..RB7 in
        movwf TR_B       ;
        clrf  INTCON     ; Disabilita interrupt
        movlw b'01000110'; Enable pull-up su porta B
        movwf OPTIO      ; Copia W in OPTION
        bcf   STAT,5     ; Seleziona SRAM banco 0
        movlw .128       ; Settaggio iniziale RTCC
        movwf RTCC       ;
        clrf  PORTA      ; Poni a zero le uscite su A
        clrf  PORTB      ; Poni a zero le uscite su B
;------------------------ main program ---------------------
MAIN    clrwdt ; Azzero watchdog
        bcf   TONO       ; Tono off
        btfsc P1         ; Testo se P1 premuto
        goto  MAIN       ; Pulsante non premuto
                         ;
LPTONO  movlw b'0100'    ; Togglo tono
        xorwf PORTA,1    ; su porta A
        movlw .30        ; Carico 30 in W
        movwf FR01       ; Carico W in FR01
LP1     clrwdt           ; Azzero watchdog
        nop              ; Nessuna operazione
        decfsz FR01      ; Decrem. FR01 e testo se zero
        goto  LP1        ; Non zero
        btfsc P1         ; Testo se P1 ancora
        goto  MAIN       ; Pulsante non premuto
        movlw b'0100'    ; Togglo tono
        xorwf PORTA,1    ; su porta A
        movlw .70        ; Carico 70 in W
        movwf FR01       ; Carico W in FR01
LP2     clrwdt           ; Azzero watchdog
        nop              ; Nessuna operazione
        decfsz FR01      ; Decrem. FR01 e testo se zero
        goto  LP2        ; Non zero
        btfsc P1         ; Testo se P1 ancora
        goto  MAIN       ; Pulsante non premuto
        goto  LPTONO     ;
;-----------------------------------------------------------
INTERP                   ; Vettore interruzioni
        retfie           ;
;-----------------------------------------------------------
    END
```

proporre successivamente: al chip è stato collegato un pulsante (P1) ed un altoparlante, utilizzando così due sole linee del controller.

L'alimentazione è sempre ricavata da un regolatore a tre terminali tipo 7805.

L'oscillatore è sempre di tipo quarzato, ma poiché non è indispensabile un'ottima precisione, potrete anche sostituirlo con uno meno costoso di tipo RC.

La soluzione adottata per il collegamento dell'altoparlante è senza dubbio la più semplice possibile, ma non offre l'elevata potenza necessaria per alimentare una tromba esponenziale oppure un piezo. Per far ciò, dovrete sostituire il transistor, che è di tipo BC337, con un amplificatore più potente.

Abbiamo volutamente applicato questa soluzione perché in poco tempo e con pochi componenti fornosce subito un'idea del suono generato dal PIC.

Il singolo tono

Passiamo allora subito a vedere il programma che permette la generazione di un singolo tono, ovvero una frequenza fissa.

Andiamo a vedere come viene implementata la frequenza di circa 1 kHz prefissata, tralasciando sia le impostazioni che la configurazione iniziale, già ampiamente illustrate nei mesi scorsi e passiamo subito alla parte di programma che genera il tono. Alla label MAIN si azzera il watchdog, poi si pone a zero l'uscita relativa all'altoparlante e si testa il pulsante P1. Fino a quando questo pulsante non viene premuto, il PIC rimane a "loopare" su queste istruzioni appena descritte.

Non appena P1 viene premuto, il controller entra nel loop di generazione tono, inizializzato con la label LPTONO.

Le prime due istruzioni consentono, come abbiamo già visto in altre occasioni, di invertire lo stato dell'uscita relativa all'altoparlante, nel nostro caso la RA2.

Così, se prima avevamo uno zero, adesso ci troviamo un uno.

PROGRAMMA 13

```
TITLE 'PROG13: Sirena bitonale per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------
RTCC    EQU 01H          ; Real Time Clock Counter
PCL     EQU 02H          ; Program Counter
STAT    EQU 03H          ; Registro di stato
PORTA   EQU 05H          ; Porta A
PORTB   EQU 06H          ; Porta B
EDATA   EQU 08H          ; Data EEPROM
EADR    EQU 09H          ; Address EEPROM
ECON1   EQU 88H          ; Stato delle operaz. in EEPROM
ECON2   EQU 89H          ; Vedi ECON1
INTCON  EQU 0BH          ; Registro abilitazione interrupt
TR_A    EQU 85H          ; Tris A
TR_B    EQU 86H          ; Tris B
OPTIO   EQU 81H          ; Registro OPTION
;-----------------------------------------------------------
FR01    EQU 0CH          ;
FR02    EQU 0DH          ;
FR03    EQU 0EH          ;
;-----------------------------------------------------------
#define  P1 PORTB,7      ; Pulsante 1
#define  TONO PORTA,2    ; Uscita segnale BF
;-----------------------------------------------------------
        ORG   0
        goto  START      ;Reset vector
        nop
        nop
        nop
        goto  INTERP     ;Interrupt vector

;-----------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------
START   bsf   STAT,5     ; Seleziona SRAM banco 1
        movlw b'0000'    ; RA out
        movwf TR_A       ;
        movlw b'11110000'; RB0..RB3 out  RB4..RB7 in
        movwf TR_B       ;
        clrf  INTCON     ; Disabilita interrupt
        movlw b'01000110'; Enable pull-up su porta B
        movwf OPTIO      ; Copia W in OPTION
        bcf   STAT,5     ; Seleziona SRAM banco 0
        movlw .128       ; Settaggio iniziale RTCC
        movwf RTCC       ;
        clrf  PORTA      ; Poni a zero le uscite su A
        clrf  PORTB      ; Poni a zero le uscite su B
;------------------- main program -------------------
MAIN    clrwdt           ; Azzero watchdog
        bcf   TONO       ; Tono off
        btfsc P1         ; Testo se P1 premuto
        goto  MAIN       ; Pulsante non premuto
                         ;
LPT2    movlw .50        ; Carico 50 in W
        movwf FR02       ; Carico W in FR02
LPT1    movlw .10        ; Carico 10 in W
        movwf FR03       ; Carico W in FR03
LPTONO  movlw b'0100'    ; Togglo tono
        xorwf PORTA,1    ; su porta A
        movlw .100       ; Carico 100 in W
        movwf  FR01      ; Carico W in FR01
LP1     clrwdt           ; Azzero watchdog
        nop              ; Nessuna operazione
        decfsz FR01      ; Decrem. FR01 e testo se zero
        goto  LP1        ; Non zero
        decfsz FR03      ; Vedo se scaduto ciclo interno
        goto  LPTONO     ;
        btfsc P1         ; Testo se P1 ancora
        goto  MAIN       ; Pulsante non premuto
        decfsz FR02      ; Vedo se scaduto ciclo principale
        goto  LPT1       ;
        movlw .100       ; Carico 100 in W
        movwf FR02       ; Carico W in FR02
LPTB    movlw .10        ; Carico 10 in W
        movwf FR03       ; Carico W in FR03
LPTB1   movlw b'0100'    ; Togglo tono
        xorwf PORTA,1    ; su porta A
        movlw .50        ; Carico 50 in W
        movwf FR01       ; Carico W in FR01
LPB1    clrwdt           ; Azzero watchdog
        nop              ; Nessuna operazione
        decfsz FR01      ; Decrem. FR01 e testo se zero
        goto  LPB1       ; Non zero
        decfsz FR03      ; Vedo se scaduto ciclo interno
        goto  LPTB1      ;
        btfsc P1         ; Testo se P1 ancora
        goto  MAIN       ; Pulsante non premuto
        decfsz FR02      ; Vedo se scaduto ciclo principale
        goto  LPTB       ;
        goto  LPT2       ;
;-----------------------------------------------------------
INTERP                   ; Vettore interruzioni
        retfie           ;
;-----------------------------------------------------------
        END
```

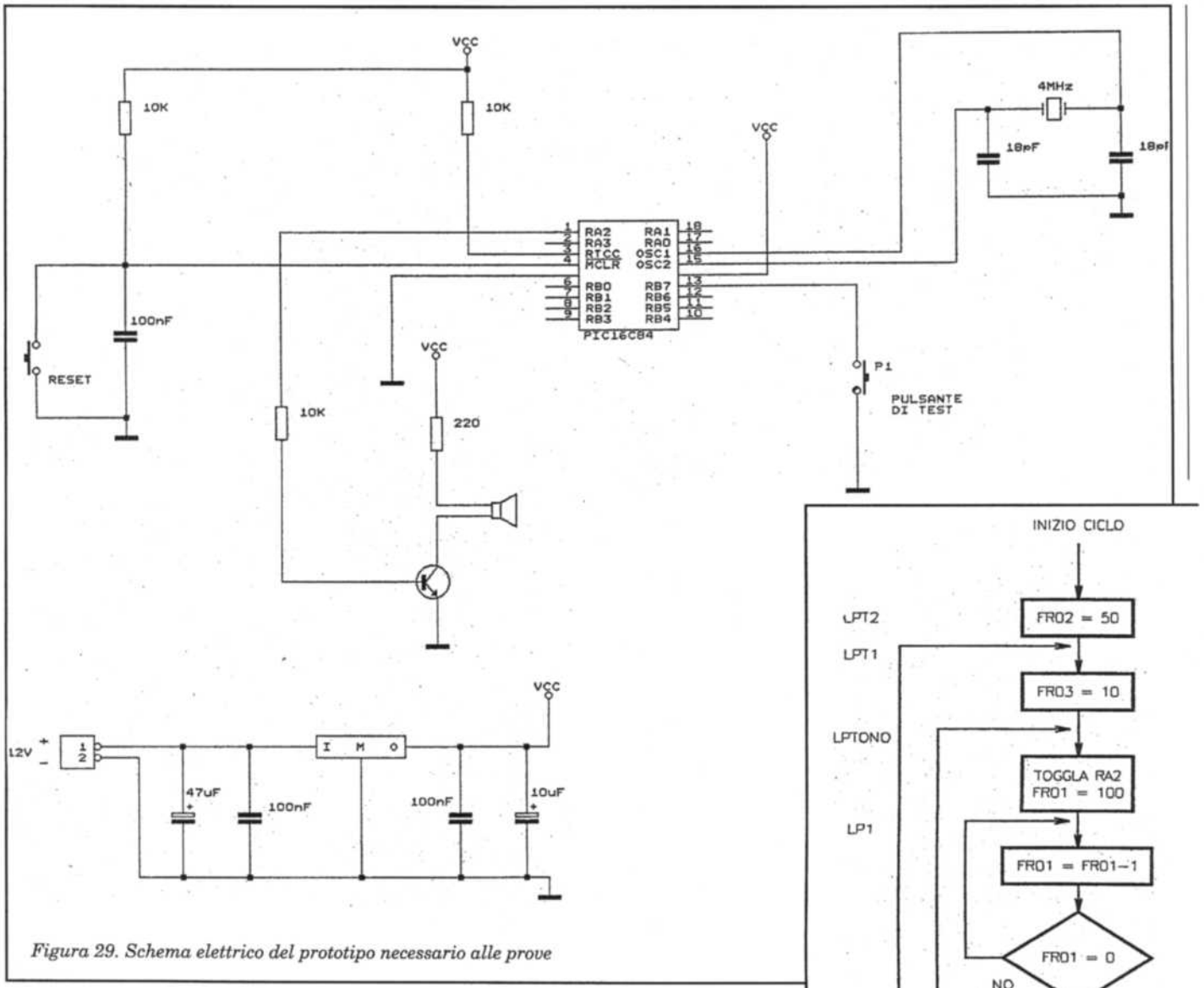


Figura 29. Schema elettrico del prototipo necessario alle prove

Poi viene caricato il registro FR01 con il valore decimale 100. A questo punto è stato introdotto un loop di ritardo che fa capo alla label LP1. Il ritardo introdotto è, come sappiamo, di 5 (numero cicli) x 1 (tempo per un ciclo in microsecondi) x 100 (numero impostato nel registro FR01), ovvero in totale 500 microsecondi. Al termine di questo ritardo, viene nuovamente testato il pulsante P1 per vedere se è sempre premuto. In caso affermativo, si torna alla label LPTONO, altrimenti si torna al ciclo iniziale.

Facendo due conti, si vede che la frequenza di uscita sul pin RA2 del PIC sarà di circa 1/(500 + 500) ovvero di 1.000 Hz. Provate a variare il valore caricato nel registro FR01 e vedrete che anche la frequenza di uscita varierà.

Quale saranno le due frequenze minima e massima che potrete ottenere da questo programma? A voi la risposta.

Tono con duty-cicle variabile

Vediamo ora come implementare una frequenza che abbia il duty-cicle non esattamente del 50%, ma variabile a piacere.

Come potete vedere dal listato di programma 12, non si sono dovute inserire molte istruzioni, ma soltanto un loop di attesa simile al precedente, con i tempi diversi.

Nel primo loop, carichiamo il registro FR01 con il valore 30, mentre nel secondo carichiamo il valore 70.

Intuitivamente allora il rapporto lavoro-pausa sarà di 30/70 e la frequenza di uscita resterà pressoché immutata.

Variando questi due tempi, è possibile ottenere la medesima frequenza con duty-cicle variabili tra 1 e 99.

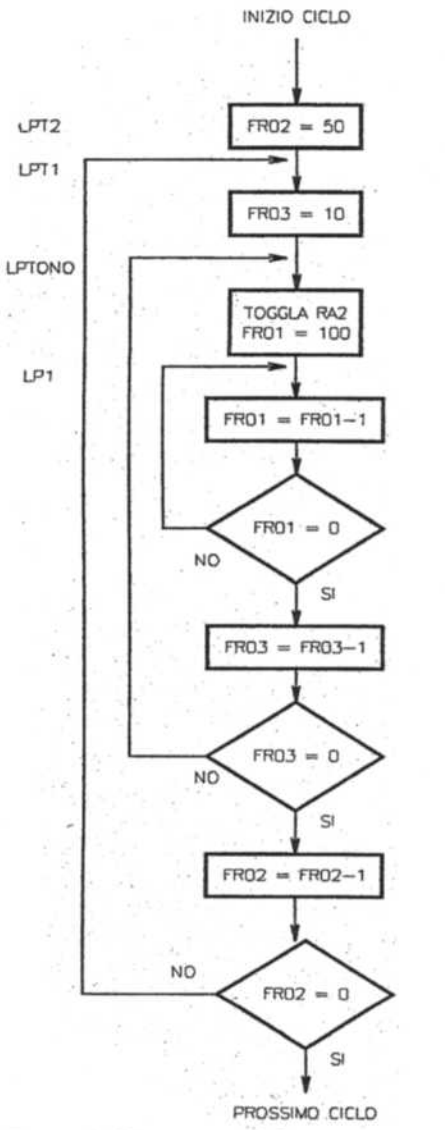


Figura 30. Diagramma di flusso del programma PROG13

PROGRAMMA 14

```
TITLE 'PROG14: Sweep per Progetto'
 list F=INHX8M,P=16C84
;-------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-------------------------------------------------------------
FR01    EQU 0CH         ;
FR02    EQU 0DH         ;
FR03    EQU 0EH         ;
FR04    EQU 0FH         ;
FR05    EQU 10H         ;
;-------------------------------------------------------------
#define P1 PORTB,7      ; Pulsante 1
#define TONO PORTA,2    ; Uscita segnale BF
;-------------------------------------------------------------
        ORG   0
        goto  START     ;Reset vector
        nop
        nop
        nop
        goto  INTERP    ;Interrupt vector
;------------------- Ritardo 500 uS --------------------------
RITARD  movlw .100      ; Carico 100 in W
        movwf FR01      ; Carico W in FR01
LP1     clrwdt          ; Azzero watchdog
        nop             ; Nessuna operazione
        decfsz FR01     ; Decrem. FR01 e testo se zero
        goto  LP1       ; Non zero
        return          ;
;-------------------------------------------------------------
; START PROGRAM
;-------------------------------------------------------------
START   bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0000'   ; RA out
        movwf TR_A      ;
        movlw b'11110000'; RB0..RB3 out  RB4..RB7 in
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        movlw b'01000110'; Enable pull-up su porta B
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf RTCC      ;
        clrf  PORTA     ; Poni a zero le uscite su A
        clrf  PORTB     ; Poni a zero le uscite su B
;------------------- main program ----------------------------
MAIN    clrwdt          ; Azzero watchdog
        bcf   TONO      ; Tono off
        btfsc P1        ; Testo se P1 premuto
        goto  MAIN      ; Pulsante non premuto
                        ;
LPTONO  movlw .100      ; Carico 100 in W
        movwf FR02      ; Carico W in FR02
        clrf  FR03      ; Azzero FR03
LPA4    movlw .10       ; Carico 10 in W
        movwf FR04      ; Carico W in FR04
        incf  FR03      ; Incremento di 1 FR03
LPA3    movlw b'0100'   ; Togglo tono
        xorwf PORTA,1   ; su porta A
        movlw .50       ; Carico 50 in W
        movwf FR05      ; Carico W in FR05
        movf  FR03,0    ; Carico FR03 in W
        subwf FR05      ; Sottraggo W da FR05
        clrwdt          ; Azzero watchdog
LPA1    btfsc P1        ; Testo se P1 sempre premuto
        goto  MAIN      ; Non piu premuto
        decfsz FR05     ; Decremento FR05 e testo se 0
        goto  LPA1      ;
        movlw b'0100'   ; Togglo tono
        xorwf PORTA,1   ; su porta A
        movlw .50       ; Carico 50 in W
        movwf FR05      ; Carico W in FR05
        movf  FR03,0    ; Carico FR03 in W
        subwf FR05      ; Sottraggo W da FR05
        clrwdt          ; Azzero watchdog
LPA2    btfsc P1        ; Testo se P1 sempre premuto
        goto  MAIN      ; Non piu premuto
        decfsz FR05     ; Decremento FR05 e testo se 0
        goto  LPA2      ; Non zero
        decfsz FR04     ; Decremento FR04 e testo se 0
        goto  LPA3      ; Non zero
        decfsz FR02     ; Decremento FR02 e testo se 0
        goto  LPA4      ; Non zero
        goto  LPTONO    ; Zero
;-------------------------------------------------------------
INTERP                  ; Vettore interruzioni
        retfie          ;
;-------------------------------------------------------------
        END
```

Il bitonale

Per coloro che necessitano di un segnale di allarme, dobbiamo proporre almeno un classico bitonale, che troviamo con il programma PROG13.

Questa volta, il listato è leggermente più complesso e, per assimilarlo, non vedremo istruzione per istruzione che cosa fa, dato che ormai siete in grado di capirlo da soli anche per mezzo dei commenti a fianco, ma spiegheremo il diagramma di flusso, in modo da poter poi comprendere i vari passaggi.

In Figura 30 vediamo il flow-chart relativo al primo ciclo del programma PROG13. In totale abbiamo due cicli, uno per ottenere la prima frequenza, l'altro per la seconda.

Esamineremo il primo, dato che sono identici eccetto il valore caricato nei tre registri.

Si creano tre loop, gestiti rispettivamente con i tre registri FR02, FR03 e FR01.

In pratica, si genera la frequenza desiderata impostata con il valore caricato in FR01 per il tempo deciso dai due valori inseriti in FR03 e FR02. Variando questi tre valori, è possibile modificare sia le due frequenze, sia le tempistiche di funzionamento.

Lo sweep

Per concludere, non poteva mancare lo sweep di frequenza, implementato con il programma PROG14. Per quest'ultimo programma, lasciamo a voi il compito di capire come viene ottenuto lo sweep, che ricordiamo altri non è che la variazione di una frequenza nel tempo. Ad esempio, si prendano le due frequenze 1.000 e 3.000 Hz. Si decida poi di partire con la frequenza più bassa e, in un tempo di 100 mS si arrivi a quella più alta.

Avremo così realizzato uno sweep da 1.000 a 3.000 Hz in 100 mS.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Ormai chi ci ha seguito nelle scorse puntate ha imparato molto sulla programmazione dei pic, ma per poter realizzare programmi complessi è necessario affinare la tecnica e l'unico modo possibile è continuare a seguire i nostri suggerimenti

Paola Sbrana - 9ª parte
su gentile concessione della Microlabs

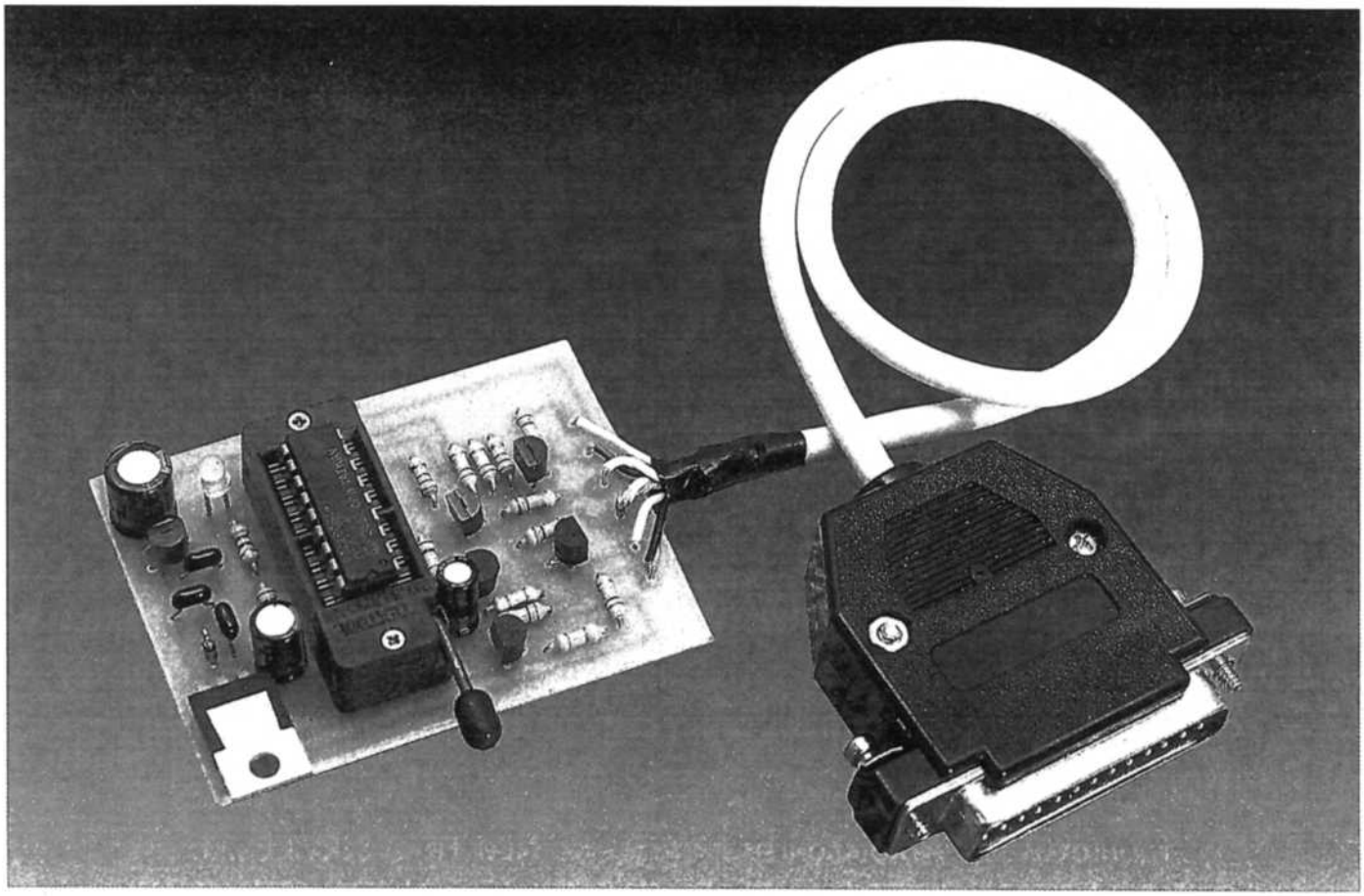
Questo mese, ci dedicheremo all'impiego dei cosiddetti "interrupt", che tanto incutono timore ai più inesperti. Ma che cosa sono questi interrupt?

Come abbiamo già visto, un programma non è altro che una sequenza di istruzioni che il controllore decodifica ed esegue. La normale logica vuole che le istruzioni vengano eseguite una dopo l'altra, in modo tale da poter gestire il funzionamento del controller in modo "passo per passo".

Ma ci sono eventi che può far comodo rilevare in qualsiasi momento dell'esecuzione di un programma, come per esempio l'arrivo di una comunicazione seriale, oppure l'overflow di un timer, oppure un fronte su di un pin. Ecco allora che entrano in gioco gli interrupt: ad un preciso evento richiesto, il controllore interrompe (da interrupt!) quello che stava facendo e passa ad una subroutine detta "subroutine di interrupt". Nel caso della seriale, tale subroutine si incaricherà di ricevere correttamente tutto il byte trasmesso dall'esterno. Al termine della subroutine di interrupt, il programma riprende dal punto in cui era stato lasciato.

Le nostre applicazioni

Le sorgenti di interrupt disponibili sul PIC16C84 sono quattro: un fronte programmabile sul pin RB0, l'overflow del timer TMR0, il cambio di stato di un



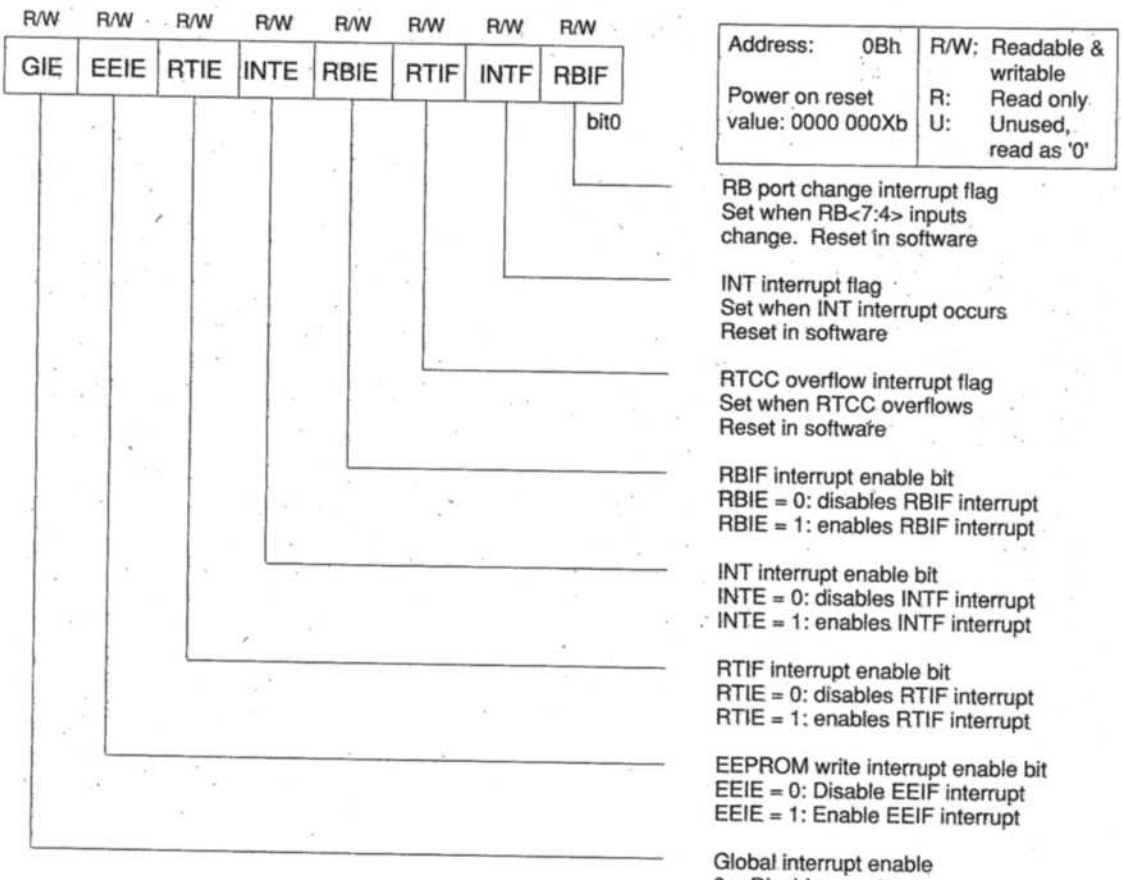


Figura 31. Configurazione del registro INTCON

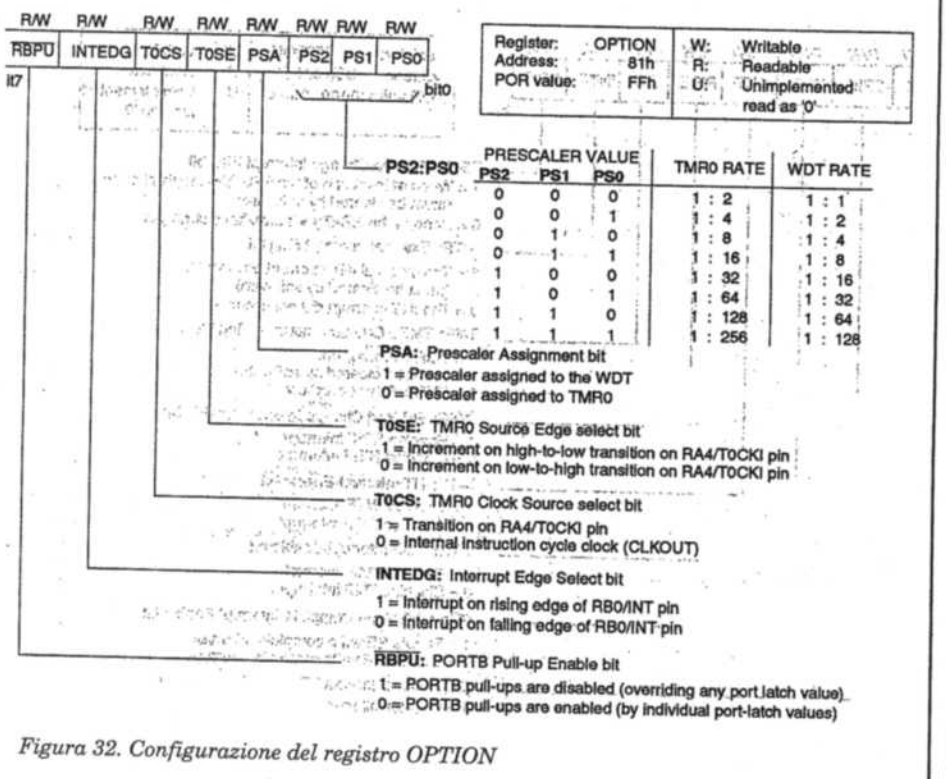


Figura 32. Configurazione del registro OPTION

pin della porta "B" da RB7 a RB4 ed il termine della fase di scrittura nella EEPROM dei dati.

Vedremo allora come si possono scrivere programmi che lavorano con interrupt, trasportabili su ogni microcontrollore della famiglia PIC16CXXX e 17CXX.

Ma prima di passare ai programmi di esempio, andiamo a vedere quali risorse hardware ci vengono messe a disposizione dal PIC per poter usufruire di questi interrupt.

In Figura 31, abbiamo il registro di gestione degli interrupt: analizziamolo in dettaglio. Il bit 0 RBIF (RB Interrupt Flag) viene settato dall'hardware quando sui pin RB4, RB5, RB6, RB7 si ha un cambio di stato ed il reset deve essere fatto via software. Il bit 1 INTF (INTerrupt Flag) viene settato dall'hardware quando avviene un interruzione per un fronte sul pin RB0. A proposito di questo interrupt, dobbiamo precisare che la programmazione del fronte avviene impostando il bit 6 (INTEDG) del registro OPTION, come si vede in Figura 32.

Anche in questo caso, il reset viene fatto via software. Il bit 2 RTIF (RTcc Interrupt Flag) viene settato anch'esso

PROGRAMMA 15 - PROVA 1

```
TITLE 'PROG15: Prova 1 interrupt RB<7..4>  per Progetto'
 list F=INHX8M,P=16C841
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ;
FR02    EQU 0DH         ;
FR03    EQU 0EH         ;
STEMP   EQU 02EH        ; Registro memorizzazione stato
WTEMP   EQU 02FH        ; Registro memorizzazione W
;-----------------------------------------------------------------
#define RP0 STAT,5      ; Flag selezione banco ram
#define RBIF INTCON,0   ; RB Interrupt Flag
#define RBIE INTCON,3   ; RB Interrupt Enable
#define GIE INTCON,7    ; Global Interrupt Enable
;-----------------------------------------------------------------
#define P1 PORTB,7      ; Pulsante 1
#define P2 PORTB,6      ; Pulsante 2
#define P3 PORTB,5      ; Pulsante 3
#define P4 PORTB,4      ; Pulsante 4
#define D1 PORTB,0      ; Led 1
#define D2 PORTB,1      ; Led 2
#define D3 PORTB,2      ; Led 3
#define D4 PORTB,3      ; Led 4
;-----------------------------------------------------------------
        ORG   0
        goto  START     ;Reset vector

        ORG   4         ;Interrupt vector
        movwf WTEMP     ;potrebbe essere in banco 0 o 1
        swapf STAT,0    ;Swap STAT in TEMP
        bcf   RP0       ;Selezione banco 0
        movwf STEMP     ;Memorizzo stato
        ;
        btfss RBIF      ;Interrupt da RB?
        goto  ENDINT    ;Uscita dalla subroutine di int.
        decfsz FR03     ;Test se FR03 = 0
        goto  ENDINT    ;per rilascio
        decfsz FR02     ;Test se FR02 = 0
        goto  RILASC    ;per rilascio
        comf  PORTB,0   ;Copia porta B complement. in W
        andlw b'11110000' ;Maschera per pulsanti
        movwf FR01      ;Copia input in FR01
        swapf FR01,0    ;Swap FR01 in W
        xorwf PORTB     ;OR-EX tra porta B e W (toggle)
        bcf   RBIF      ;Reset RBIF
        movlw .10       ;Carico 10 in W
        movwf  FR02     ;Copio W in FR02
RILASC  movlw .100      ;Carico 100 in W
        movwf FR03      ;Copio W in FR03
        ;
ENDINT  bsf   GIE       ;Abilita interrupt RB
        bcf   RP0       ;
        swapf STEMP,0   ;Ripristino stato
        movwf STAT      ;
        swapf WTEMP,1   ;Ripristino W
        swapf WTEMP,0   ;
        retfie          ;
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   clrf  PORTB     ; Azzera uscite su porta B
        bsf   STAT,5    ; Seleziona SRAM banco 1
        movlw b'0000'   ; RA out
        movwf TR_A      ;
        movlw b'11110000' ; RB0..RB3 out  RB4..RB7 in
        movwf TR_B      ;
        clrf  INTCON    ; Disabilita interrupt
        bsf   RBIE      ; Abilita RB Interrupt
        bsf   GIE       ; Abilita Global Interrupt
        movlw b'01000110' ; Prescaler 1:128
        movwf OPTIO     ; Copia W in OPTION
        bcf   STAT,5    ; Seleziona SRAM banco 0
        movlw .128      ; Settaggio iniziale RTCC
        movwf RTCC      ;
        movlw b'0001'   ;
        movwf  PORTA    ;
;-------------------- main program -------------------------------
MAIN    clrwdt          ;
        goto  MAIN      ;
;-----------------------------------------------------------------
        END
```

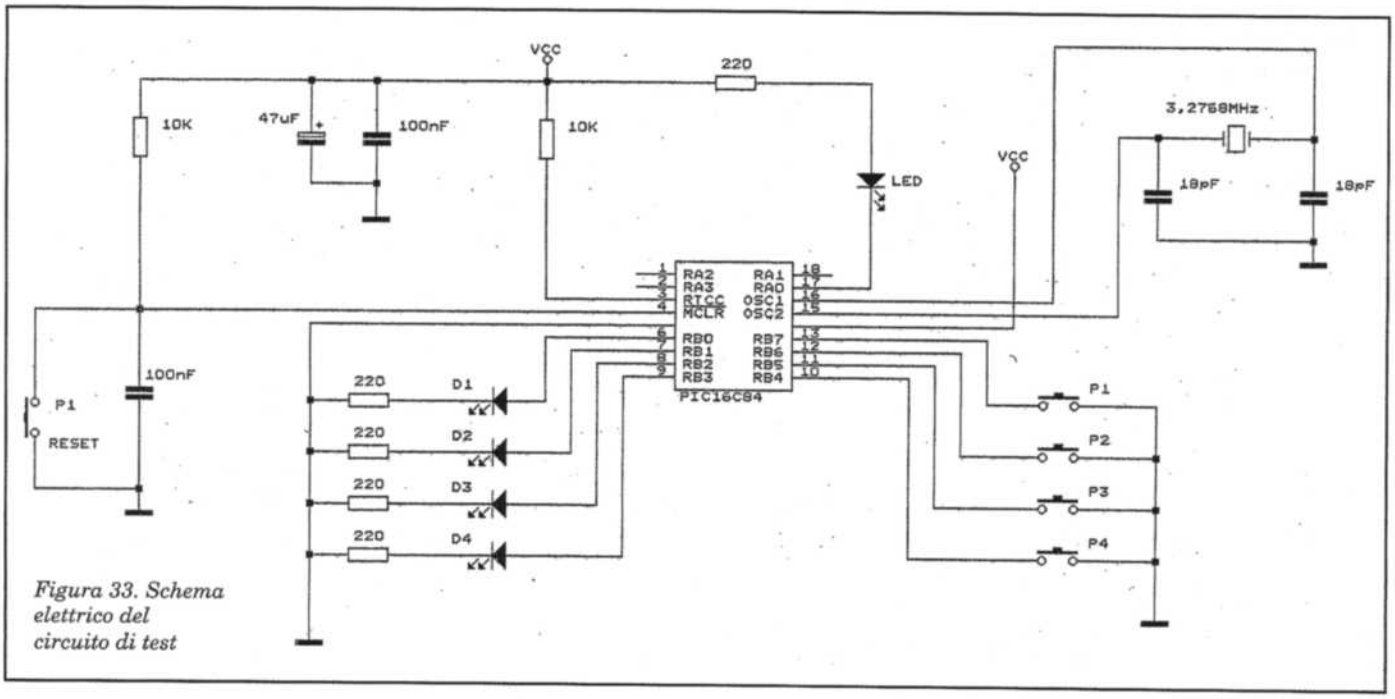


Figura 33. Schema elettrico del circuito di test

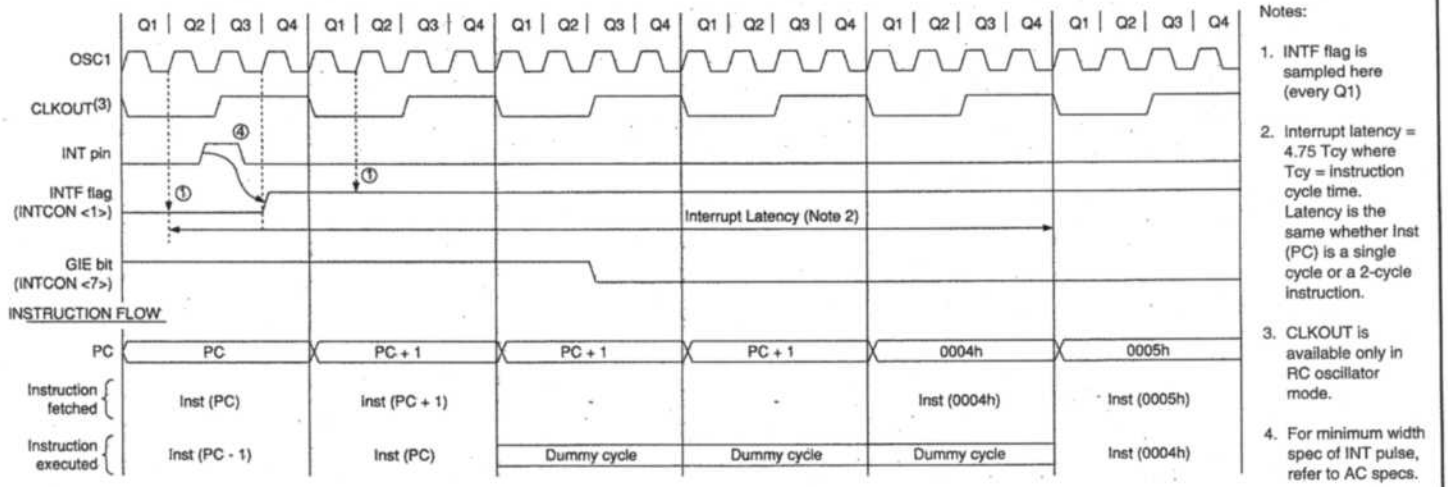


Figura 34. Timing sull'interrupt da INT

dall'hardware quando il timer TMR0 va in overflow.

Il reset deve sempre avvenire via software. Il bit 3, RBIE, è quello che abilita o meno il generarsi di un interrupt sul cambio di stato della porta RB<7..4>. Il bit 4, INTE, abilita o meno il generarsi di un interrupt ad ogni fronte sul pin RB0. Il bit 5, RTIE, abilita o meno un interrupt dal timer TMR0, mentre il bit 6, EEIE, abilita o meno il generarsi di un interrupt al termine di una eventuale scrittura sulla EEPROM dei dati. Infine, il bit 7, GIE, (Global Interrupt Enable) abilita o meno tutte le precedenti sorgenti di interrupt.

Lo schema elettrico del prototipo necessario alle nostre prove è quello di Figura 33. Per come sono stati configurati i pin, si vede subito che la sorgente di interrupt relativa al fronte sul pin RB0 non sarà disponibile, in quanto tale pin viene impiegato come uscita. Sfrutteremo invece l'interrupt della porta RB<7..4> e quello dell'overflow del timer. Vediamo allora il primo programma: PROG 15.

Con questo sorgente, abilitiamo il solo interrupt relativo alla variazione di stato della porta RB sui pin da 7 a 4. Per prima cosa si abilita l'RBIE e, successivamente, il GIE. Ciò implica che, ogni volta che premiamo un pulsante, verrà generato un interrupt di tipo RBIF, settando il bit corrispondente e saltando al vettore degli interrupt dopo ORG 4.

Le prime operazioni da eseguire sono il salvataggio del registro W e del registro di stato, perché al termine della subroutine di interrupt sarà necessario riaverli con gli stessi valori contenuti prima dell'interrupt. Poi si va a testare se l'interrupt è stato generato proprio dalla variazione della porta RB<7..4> ed infine si passa alla routine vera e propria di gestione dell'evento.

Qui abbiamo implementato un ritardo di alcuni millisecondi, prima di togglare il Led corrispondente al pulsante premuto, per il solito problema del rimbalzo sul pulsante. Al termine della gestione dell'evento, passiamo al ripristi-

PROGRAMMA 15 - PROVA 2

```
TITLE 'PROG15: Prova 2 interrupt RB<7..4> e RTCC per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H          ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ;
FR02    EQU 0DH         ;
FR03    EQU 0EH         ;
FR04    EQU 0FH         ;
STEMP   EQU 02EH        ; Registro memorizzazione stato
WTEMP   EQU 02FH        ; Registro memorizzazione W
;-----------------------------------------------------------------
#define RP0 STAT,5      ; Flag selezione banco ram
#define RBIF INTCON,0   ; RB Interrupt Flag
#define RBIE INTCON,3   ; RB Interrupt Enable
#define RTIF INTCON,2   ; RTCC Interrupt Flag
#define RTIEINTCON,5    ; RTCC Interrupt Enable
#define GIE INTCON,7    ; Global Interrupt Enable
;-----------------------------------------------------------------
#define P1PORTB,7       ; Pulsante 1
#define P2 PORTB,6      ; Pulsante 2
#define P3 PORTB,5      ; Pulsante 3
#define P4 PORTB,4      ; Pulsante 4
#define D1 PORTB,0      ; Led 1
#define D2 PORTB,1      ; Led 2
#define D3 PORTB,2      ; Led 3
#define D4 PORTB,3      ; Led 4
;-----------------------------------------------------------------
        ORG     0
        goto    START   ;Reset vector

        ORG     4       ;Interrupt vector
        movwf   WTEMP   ;potrebbe essere in banco 0 o 1
        swapf   STAT,0  ;Swap STAT in TEMP
        bcf     RP0     ;Selezione banco 0
        movwf   STEMP   ;Memorizzo stato
                        ;
        btfss   RTIF    ;Interrupt da RTCC?
        goto    INTRB   ;
        bcf     RTIF    ;Reset RTIF
        decfsz  FR04    ;Test se FR04 = 0
        goto    INTRB   ;per toggle led
        movlw   b'0001' ;Copia 1 in W
        xorwf   PORTA   ;Toggla led
        movlw   .3      ;Carico 3 in W
        movwf   FR04    ;Copio W in FR04
                        ;
INTRB   btfss   RBIF    ;Interrupt da RB?
        goto    ENDRB   ;Uscita dalla subroutine di int.
        decfsz  FR03    ;Test se FR03 = 0
        goto    ENDRB   ;per rilascio
        decfsz  FR02    ;Test se FR02 = 0
        goto    RILASC  ;per rilascio
        comf    PORTB,0 ;Copia porta B complement. in W
        andlw   b'11110000' ;Maschera per pulsanti
        movwf   FR01    ;Copia input in FR01
        swapf   FR01,0  ;Swap FR01 in W
        xorwf   PORTB   ;OR-EX tra porta B e W (toggle)
        bcf     RBIF    ;Reset RBIF
        movlw   .10     ;Carico 10 in W
        movwf   FR02    ;Copio W in FR02
RILASC  movlw   .10     ;Carico 10 in W
        movwf   FR03    ;Copio W in FR03
                        ;
ENDRB   bsf     GIE     ;Abilita interrupt
        bsf     RBIE    ;Abilita RB Interrupt
        bsf     RTIE    ;Abilita interrupt RTCC
        bcf     RP0     ;
        swapf   STEMP,0 ;Ripristino stato
        movwf   STAT    ;
        swapf   WTEMP,1 ;Ripristino W
        swapf   WTEMP,0 ;
        retfie          ;
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   clrf    PORTB   ; Azzera uscite su porta B
        bsf     STAT,5  ; Seleziona SRAM banco 1
        movlw   b'0000' ; RA out
        movwf   TR_A    ;
        movlw   b'11110000' ; RB0..RB3 out  RB4..RB7 in
        movwf   TR_B    ;
        clrf    INTCON  ; Disabilita interrupt
        movlw   b'01000111' ; Prescaler 1:256 al RTCC
        movwf   OPTIO   ; Copia W in OPTION
        bsf     RBIE    ; Abilita RB Interrupt
        bsf     RTIE    ; Abilita RTCC interrupt
        bsf     GIE     ; Abilita Global Interrupt
        bcf     STAT,5  ; Seleziona SRAM banco 0
        movlw   .128    ; Settaggio iniziale RTCC
        movwf   RTCC    ;
        movlw   b'0001' ;
        movwf   PORTA   ;
        movlw   .3      ;Carico 3 in W
        movwf   FR04    ;Copio W in FR04
;-------------------- main program -------------------------------
MAIN    clrwdt          ;
        goto    MAIN    ;
;-----------------------------------------------------------------
        END
```

no del registro di stato e del registro W.

Come potete osservare, il ciclo MAIN è vuoto, ossia il programma "looppa" in continuazione sulle due istruzioni CLRWDT e GOTO MAIN. Non appena un pulsante viene premuto, l'interrupt relativo si attiva e si passa alla sua gestione. In questo modo, è possibile inserire nel loop del MAIN il programma vero e proprio, senza più curarsi della gestione dei pulsanti.

Passiamo adesso ad una gestione più complessa, che prevede l'utilizzo di due interrupt.

Rispetto al precedente, questo programma ha attivato anche l'interrupt dell'overflow del timer (RTCC o TMR0) che impiega per far lampeggiare un Led collegato sul pin RA0. La procedura di gestione è la stessa: non appena si genera un interrupt, si passa al vettore di interrupt, si salva lo stato e W e si va a gestire l'interrupt che ha causato l'interruzione. In questo caso potrebbe anche verificarsi il caso in cui si abbia generazione di interrupt da entrambe le sorgenti (timer e pulsante). Non ci sono problemi nel gestire questi casi, basta procedere con un evento alla volta.

In questo esempio, il primo interrupt che valutiamo è quello del timer, poi si passa allo stato della porta RB<7..4>.

Al termine si ripristinano i registri W e di stato e si torna al ciclo principale.

Noterete che, anche premendo i tasti casualmente, la frequenza del lampeggio del Led non subirà variazioni di alcun tipo. Una cosa da tener ben presente quando si lavora con gli interrupt, è il tempo che trascorre dal verificarsi dell'evento generatore di interrupt.

In Figura 34 troviamo il diagramma temporale relativo all'interrupt sul pin RB0. Si nota chiaramente che il tempo di latenza è di tre operazioni, mentre il tempo minimo per entrare in una routine di interrupt e riuscire, è dato dalla somma delle istruzioni che servono per salvare lo stato, ripristinarlo e poi gestire l'interrupt stesso.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Ormai è un dato di fatto: il nostro corso sta facendo scuola e viene adottato dai principali laboratori, oltre che dagli hobbisti, che si stanno avvicinando al mondo della programmazione dei componenti. Continuiamo quindi a passo spedito per svelarvi i segreti più nascosti dei Pic

Paolo Sbrana - 10ª parte
su gentile concessione della Microlabs

Le routine che vedremo questo mese sono sempre state le più richieste da tutti i lettori, sia perché di non facile gestione, sia perché da abbinare a routine di gestione di un personal computer: parleremo di interfacciamento RS-232 tra PIC e porta seriale del PC.

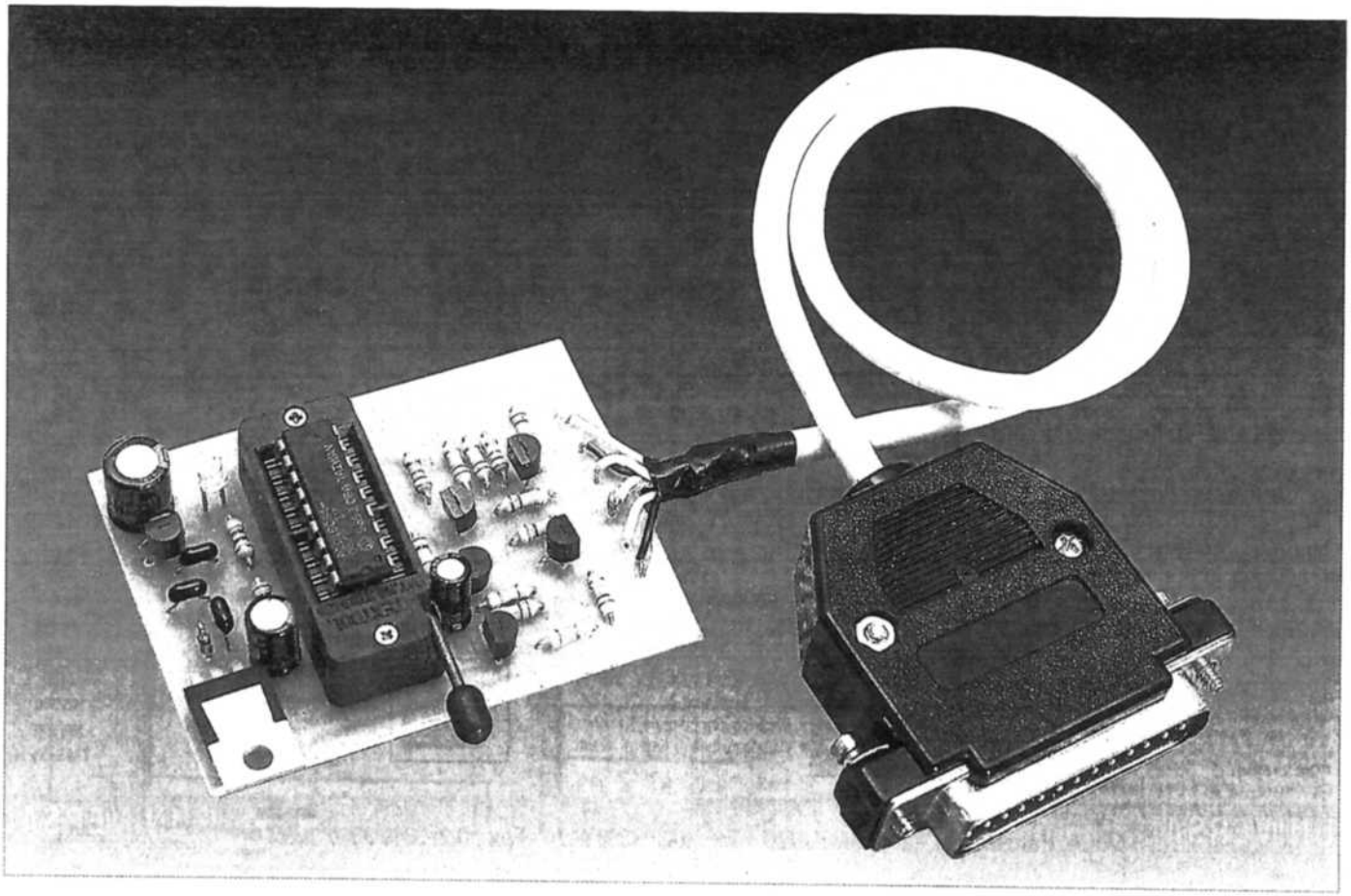
Andando in un negozio di computer, notiamo che le periferiche che vengono collegate alla presa seriale del computer sono tantissime, anche perché fino a poco tempo fa era la soluzione migliore per dialogare con il personal. Con l'avvento dei portatili poi, hanno iniziato a crearsi uno spazio anche le schede PCMCIA, ma, almeno per ora, sono ad uso esclusivo dei portatili e di pochi PC che le installano solo per una compatibilità con periferiche che già possiedono.

Chiaramente i due tipi di connessione offrono entrambi vantaggi e svantaggi.

Lo svantaggio più grande delle schede PCMCIA è quello del costo, mentre quello della RS-232 è la lentezza di trasferimento. In condizioni normali si arriva ad un massimo di 9.600 bit al secondo.

Per questi motivi, la scelta della migliore connessione tra una periferica hardware ed un computer cade nella maggior parte dei casi sul collegamento seriale RS-232.

Il PIC si adatta benissimo a dialogare con tale standard, ed a tale scopo ricordiamo che ci sono alcuni PIC che hanno già incorporata una periferica che consente di gestire tale collegamento senza scrivere righe di software.



PROGRAMMA 17

```
TITLE 'PROG17: Trasmissione seriale RS232 per Progetto'
 list F=INHX8M,P=16C84
;
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;
FR01    EQU 0CH         ; Ritardo 416uS
FR02    EQU 0DH         ; Numero bit da trasmettere
TXREG   EQU 0EH         ; Byte da trasmettere
;
#define  RP0   STAT,5   ; Flag selezione banco ram
;
#define  P1    PORTB,7  ; Pulsante 1
#define  RX    PORTB,4  ; Pin di ricezione seriale
#define  TX    PORTB,0  ; Pin di trasmissione seriale
#define  LED   PORTA,0  ; Led
;
        ORG   0
        goto  START     ;Reset vector

        ORG   4         ;Interrupt vector
        retfie  ;
;
; Routine ritardo per la seriale 1200 baud con 3,579545 MHz
;
DELARS  movlw  89       ; Scrivi 89 in W
        movwf  FR01     ; copia W in FR01
LOOPD   clrwdt          ; Resetta watchdog
        decfsz FR01     ;Decrementa FR01 e salta se zero
        goto   LOOPD    ; Vai a LOOPD
        return          ; Fine subroutine
;
; Routine di trasmissione RS232  1 start  8 data  1 stop bit
; registro TXREG il byte da trasmettere
;
TXRS    movlw  .8       ; Scrivi 8 in W
        movwf  FR02     ; Copia W in FR02
        bcf    TX       ; Bit start
        call   DELARS   ; Attesa 416uS
        call   DELARS   ; Attesa 416uS
LOOPTX  rrf    TXREG    ; Shifta TXREG a destra
        btfsc  STAT,0   ; Salta se carry = 0
        bsf    TX       ; TX pin = 1
        btfss  STAT,0   ; Salta se carry = 1
        bcf    TX       ; Tx pin = 0
        call   DELARS   ; Attesa 416uS
        call   DELARS   ; Attesa 416uS
        decfsz FR02     ; Decrementa FR02 e salta se zero
        goto   LOOPTX   ; Vai a LOOPTX
        bsf    TX       ; Stop bit
        call   DELARS   ; Attesa 416uS
        call   DELARS   ; Attesa 416uS
        return          ; Fine subroutine
;
; START PROGRAM
;
START   movlw  b'11110001' ; TX = 1
        movwf  PORTB    ;
        bsf    STAT,5   ; Seleziona SRAM banco 1
        movlw  b'0000'  ; RA out
        movwf  TR_A     ;
        movlw  b'10010000' ; RB4,RB7 in  altre out
        movwf  TR_B     ;
        clrf   INTCON   ; Disabilita interrupt
        movlw  b'01000111' ; Prescaler 1:256 al RTCC
        movwf  OPTIO    ; Copia W in OPTION
        bcf    STAT,5   ; Seleziona SRAM banco 0
        movlw  .128     ; Settaggio iniziale RTCC
        movwf  RTCC     ;
        bsf    LED      ; Led off
;-------------- main program --------------
MAIN    clrwdt          ; Resetta watchdog
        btfsc  P1       ; Testa pressione P1
        goto   MAIN     ; Vai a MAIN
        movlw  'T'      ; Scrivi il codice ascii di T in W
        movwf  TXREG    ; Copia W in TXREG
        call   TXRS     ; Invia dato RS232
RIL     clrwdt          ; Azzera watchdog
        btfss  P1       ; Attesa rilascio pulsante
        goto   RIL      ; Vai a RIL
        goto   MAIN     ; Vai a MAIN
;
        END
```

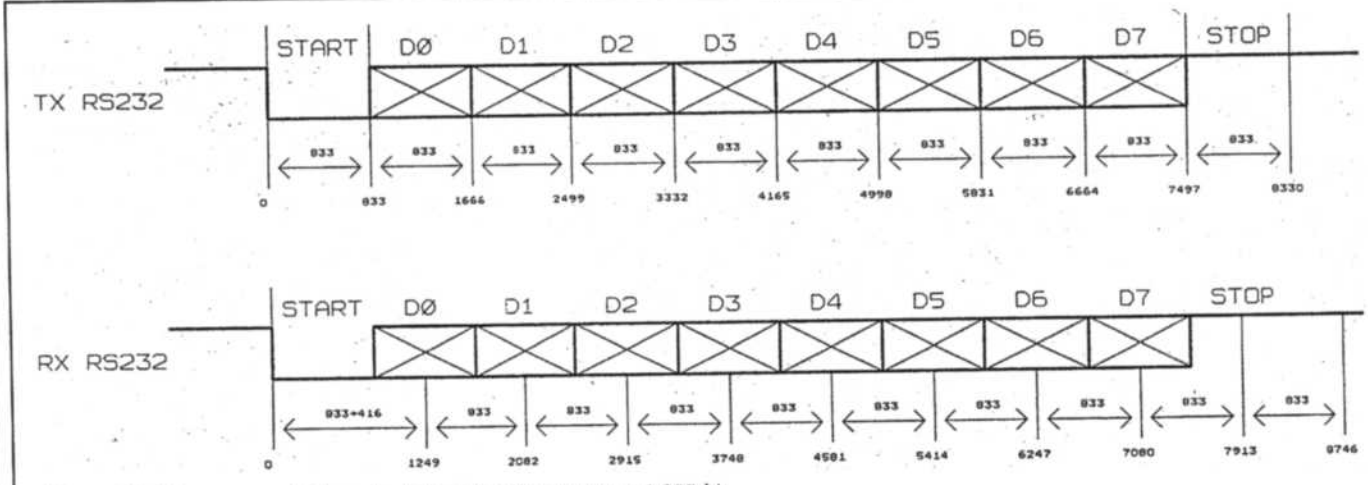


Figura 35. Diagramma dei tempi relativo al collegamento a 1.200 bit

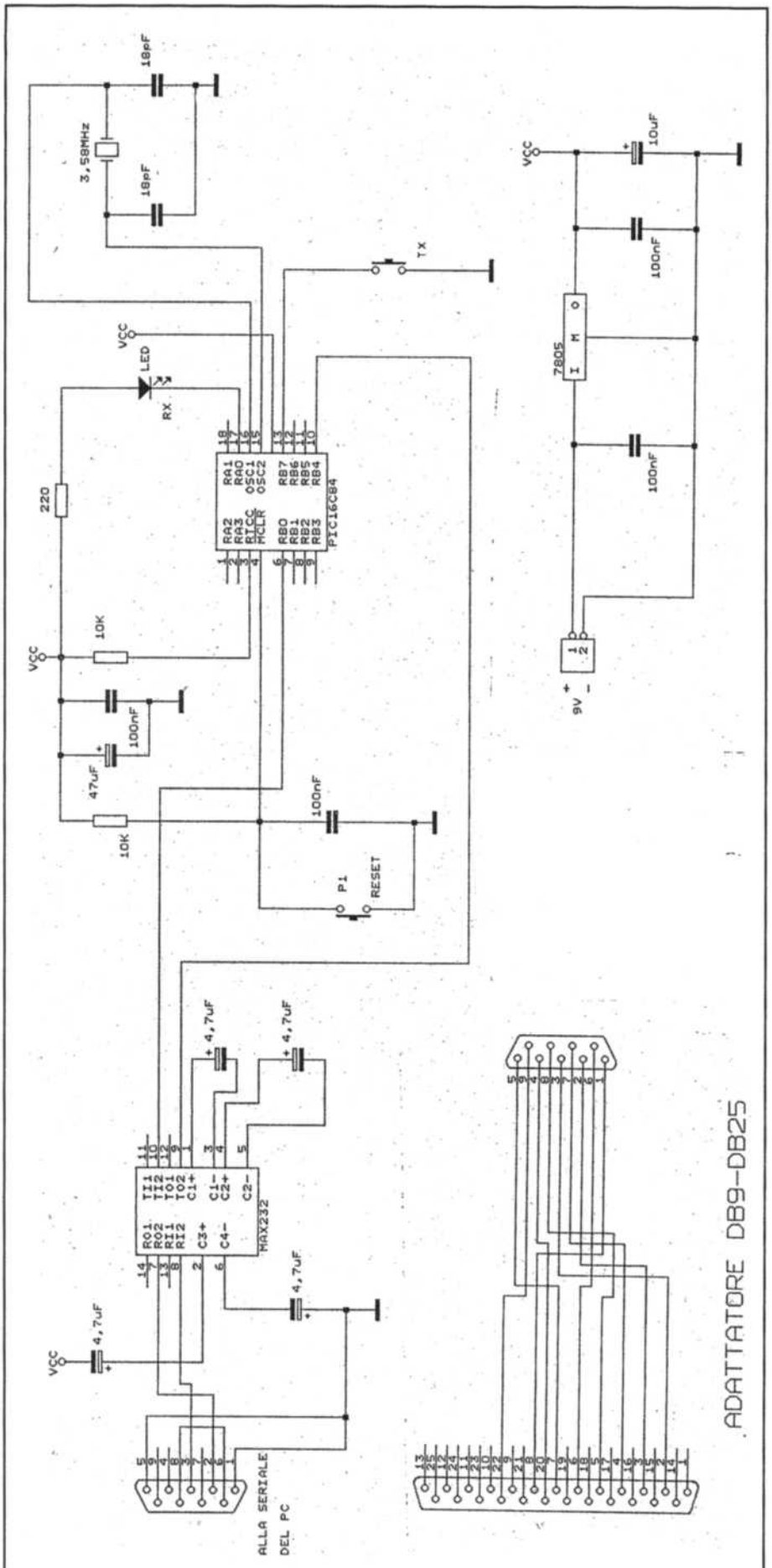


Figura 36. Schema elettrico del circuito di prova

Questi PIC però non sono programmabili con il programmatore presentato ed in più non sono realizzati in tecnologia EEPROM, quindi non li vedremo. Coloro che fossero interessati a queste informazioni, potranno richiederle direttamente alla Microchip Italia, 039/6899939.

La porta seriale

Anche se questo argomento è già stato ampliamente trattato nel mese di agosto, riprendiamo le nozioni fondamentali del collegamento RS-232.

In Figura 35 troviamo il diagramma temporale relativo ad un collegamento a 1.200 bit con modalità 8N1.

Cerchiamo di capire che cosa significa: la linea seriale è per default ad alto livello logico (tralasciamo per adesso i livelli di lavoro effettivi) ed una trasmissione si identifica con il portare a basso livello tale linea.

Vediamo allora la sezione di trasmissione. Poiché la seriale RS-232 è per definizione asincrona, cioè non legata ad alcun clock di riferimento, il trasmittente ed il ricevente devono riuscire a dialogare in un altro modo. La soluzione scelta è quella di stabilire a priori una determinata velocità di trasmissione ed un ben preciso protocollo.

Le velocità più comunemente impiegate sono i 1.200, 2.400, 4.800, 9.600 e 19.200 baud, ovvero bit al secondo.

Se facciamo un rapido calcolo, possiamo subito vedere quanto deve "durare" un bit per una determinata velocità: ad esempio per dialogare a 9.600 bit al secondo, significa che un bit deve durare 1/9.600=0,0001042 ovvero circa 104 microsecondi.

Nel nostro diagramma, abbiamo preso come riferimento una velocità di 1.200 baud.

Una volta decisa la velocità di trasferimento, due apparecchi che vogliono dialogare devono stabilire un protocollo di dialogo.

Lo standard RS-232 prevede che sia presente uno START BIT, poi vengano inviati i bit dei dati (da 5 a 8), poi è possibile inserire un bit che controlli la parità, ovvero se il numero di 1 è pari o dispari (parità pari o dispari) ed infine venga rispettato uno stop che può durare da uno a due bit (anche 1,5 nel caso di dati a 5 bit).

La trasmissione più comunemente scelta è quella detta 1.200 8N1, cioè a 1.200 bit al secondo, 8 bit di dati, nessun bit di parità ed un bit di stop.

PROGRAMMA 18

```
TITLE 'PROG18: Ricezione seriale RS232 per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ; Ritardo 416uS
FR02    EQU 0DH         ; Numero bit da trasmettere
RXREG   EQU 0EH         ; Byte da trasmettere
;-----------------------------------------------------------------
#define RP0   STAT,5    ; Flag selezione banco ram
;-----------------------------------------------------------------
#define P1    PORTB,7   ; Pulsante 1
#define RX    PORTB,4   ; Pin di ricezione seriale
#define TX    PORTB,0   ; Pin di trasmissione seriale
#define LED   PORTA,0   ; Led
;-----------------------------------------------------------------
      ORG   0
      goto  START       ;Reset vector

      ORG   4           ;Interrupt vector
      retfie            ;
;-----------------------------------------------------------------
; Routine ritardo per la seriale 1200 baud con 3,579545 MHz
;-----------------------------------------------------------------
DELARS  movlw .89         ; Scrivi 89 in W
        movwf FR01        ; copia W in FR01
LOOPD   clrwdt ; Resetta watchdog
        decfsz FR01       ; Decrementa FR01 e salta se zero
        goto  LOOPD       ; Vai a LOOPD
        return ; Fine subroutine
;-----------------------------------------------------------------
; Routine di ricezione RS232      1 start  8 data  1 stop bit
; registro RXREG il byte da ricevere
;-----------------------------------------------------------------
RXRS    call  DELARS      ; Attesa 416uS (Start bit)
        call  DELARS      ; Attesa 416uS
        movlw .8          ; Scrivi 8 in W
        movwf FR02        ; Copia W in FR02
        clrf  RXREG       ; Azzera RXREG
LOOPRX  call  DELARS      ; Attesa 416uS (Mezzo bit)
        bcf   STAT,0      ; Azzera carry
        rrf   RXREG       ; Shifta RXREG a destra
        btfsc RX          ; Testa RX pin e salta se = 0
        bsf   RXREG,7     ; Bit 7 di RXREG = 1
        call  DELARS      ; Attesa 416uS (Mezzo bit)
        decfsz FR02       ; Decrementa FR02 e salta se zero
        goto  LOOPRX      ; Vai a LOOPRX
        call  DELARS      ; Attesa 416uS (Stop bit)
        return            ; Fine subroutine
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   movlw b'11110001' ; TX = 1
        movwf PORTB       ;
        bsf   STAT,5      ; Seleziona SRAM banco 1
        movlw b'0000'     ; RA out
        movwf TR_A        ;
        movlw b'10010000' ; RB4,RB7 in  altre out
        movwf TR_B        ;
        clrf  INTCON      ; Disabilita interrupt
        movlw b'01000111' ; Prescaler 1:256 al RTCC
        movwf OPTIO       ; Copia W in OPTION
        bcf   STAT,5      ; Seleziona SRAM banco 0
        movlw .128        ; Settaggio iniziale RTCC
        movwf RTCC        ;
        bsf   LED         ; Led off
;-------------------- main program -------------------------------
MAIN    clrwdt            ; Resetta watchdog
        btfsc RX          ; Testa seriale
        goto  MAIN        ; Vai a MAIN
        call  RXRS        ; Ricevi dato RS232
        movlw 'T'         ; Scrivi T in W
        subwf RXREG,0     ; Sottrai W a RXREG
        btfsc STAT,2      ; Controllo se zero
        bcf   LED         ; Led on
        goto  MAIN        ; Vai a MAIN
;-----------------------------------------------------------------
    END
```

Per questo motivo le routine che vedremo tra poco implementano questo protocollo, ma da queste, modificando alcuni parametri, potrete ottenere il protocollo che desiderate, limitatamente alla velocità di clock del PIC.

In Figura 36 troviamo lo schema elettrico del circuito da realizzare per collaudare le routine che scriveremo. Come avrete notato, è stato inserito un MAX232 che serve ad adattare i livelli del PIC ai livelli richiesti dalla porta seriale del computer, che possono variare tra 3 e 15 volt positivi o negativi in funzione del valore del bit.

È stato inserito il pulsante TX per inviare un dato al computer e un Led RX per visualizzare se un dato giunge correttamente dal PC.

Inoltre, abbiamo riportato un adattatore tra presa a 9 poli e presa a 25 poli.

Il primo programma

Il programma che vediamo per primo è quello che ci permette di trasmettere dal PIC al PC un dato di otto bit.

Per capire come funziona, riferiamoci nuovamente alla già citata Figura 35: quando la trasmissione inizia, porta a basso livello la linea seriale e si mantiene così per tutta la durata del bit di start, che nel nostro caso è circa 833 microsecondi.

Poi si devono inviare gli otto bit dei dati, ed infine il bit di stop.

In Figura 37-a abbiamo il diagramma di flusso del programma PROG17 che, come vediamo, rispecchia quanto richiesto dal protocollo.

Al nostro programma abbiamo aggiunto il rilevamento della pressione del pulsante, per far trasmettere il carattere solo al momento desiderato.

Il carattere inviato è la "T", ma potrete cambiarlo con un qualsiasi carattere ASCII. Ovviamente, per la prova di questo programma, dovrete avere anche un gestore di porta seriale sul PC, come Telix, Procomm, Terminale ecc oppure anche un prodotto realizzato su misura.

Certo è che, per sfruttare poi la periferica che realizzerete con la seriale RS-232, dovrete anche scrivere del software di gestione sul computer.

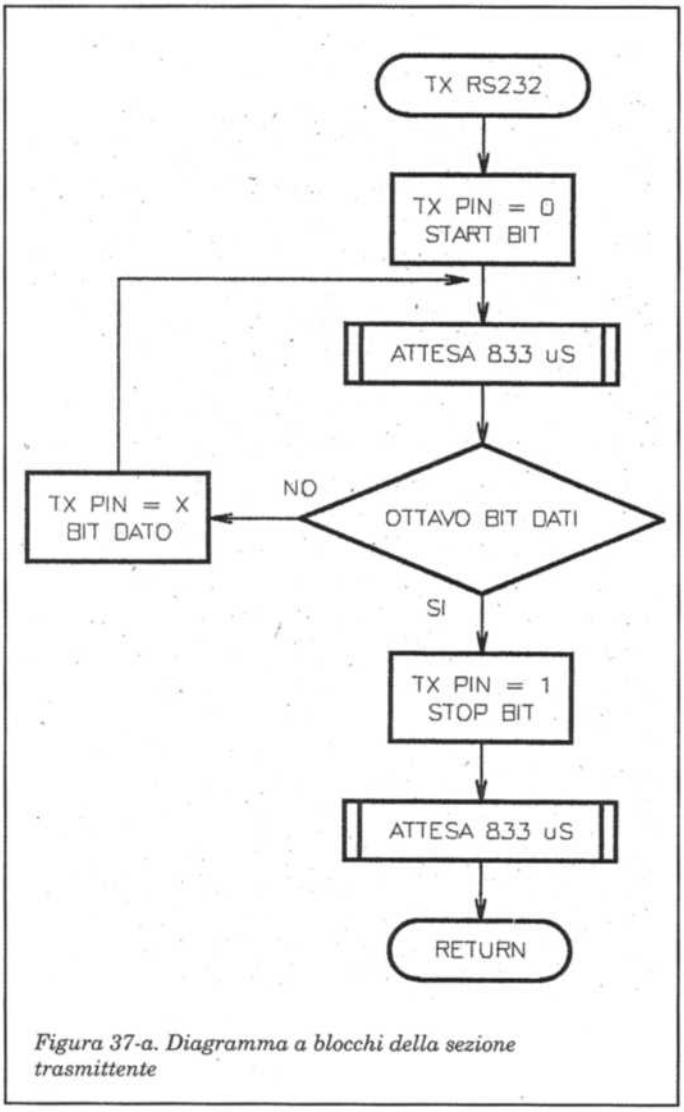


Figura 37-a. Diagramma a blocchi della sezione trasmittente

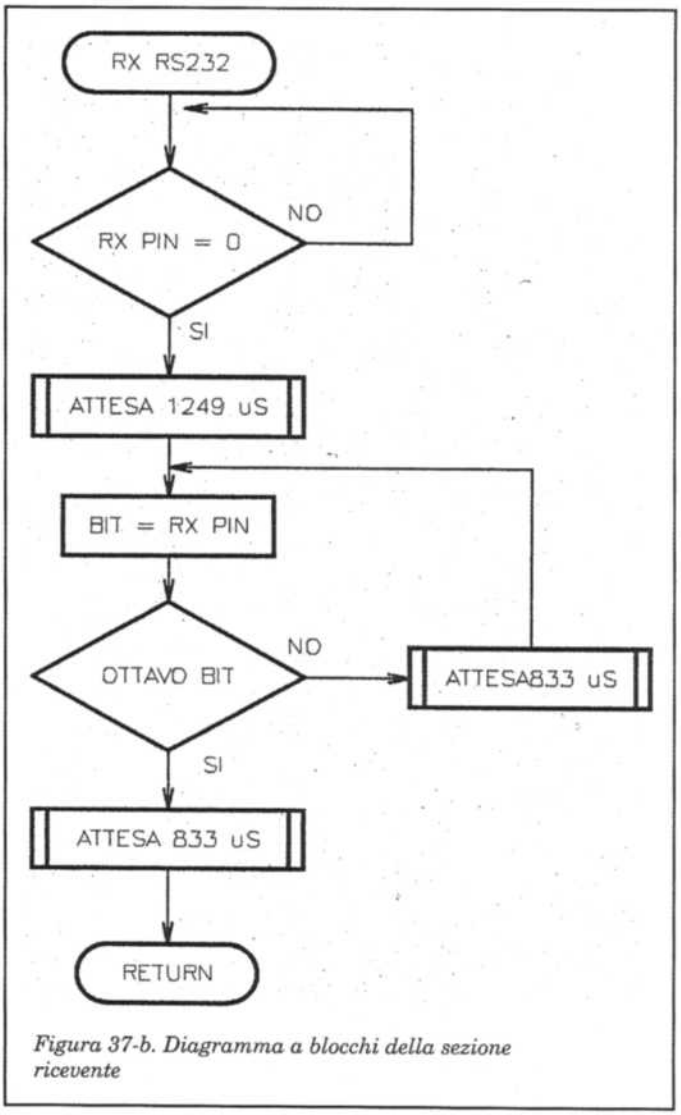


Figura 37-b. Diagramma a blocchi della sezione ricevente

Non ci sono linguaggi preferenziali, ma possiamo consigliare il glorioso BASIC, oppure il più trasportabile C, oppure l'ottimo Visual Basic, che offre tutte le potenzialità di Windows con pochissime righe di programma.

Il secondo programma

Il programma che andiamo a vedere per secondo ci consentirà di ricevere un dato dalla porta seriale del PC e di far accendere un Led se questo corrisponde al carattere da noi voluto.

Come vedete, la routine di ricezione non differisce molto da quella di trasmissione: invece che pilotare un'uscita la andiamo a leggere.

Che cosa cambia invece, è la tempistica impiegata: sempre in Figura 35, vediamo che, per analizzare correttamente il primo bit, dobbiamo attendere il tempo di un bit e mezzo a partire dalla ricezione del bit di start. Nel diagramma di Figura 37-b vediamo quale sia la procedura corretta per ricevere un dato dalla seriale: quando giunge il bit di start, si attende che sia terminato (833 µS) più si attende che sia già arrivato mezzo bit successivo (416 µS).

Poi si passa al test della linea per vedere se il bit vale 0 o 1.

Se il bit in esame è l'ottavo, si attendono altri 416 µS (bit di stop) e si torna al programma principale, altrimenti si attendono 833 µS e si passa al bit successivo.

In questo modo si possono ricevere dati con un PIC che non disponga della periferica seriale a bordo.

In questo ultimo programma, il Led si accende solo se al PIC giunge il carattere "T", che, come prima potrà poi essere modificato.

Dicevamo all'inizio che è possibile variare il protocollo impiegato variando alcune righe.

Per modificare il numero di bit da ricevere o da trasmettere, è sufficiente cambiare il valore della costante con cui viene caricato, ad inizio routine, il registro FR02.

Per modificare la velocità di trasferimento, basta modificare il valore della costante con cui viene caricato il registro FR01 nella subroutine DELARS.

Per aumentare il numero dei bit di stop, basta aggiungere a fine subroutine (sia di trasmissione che di ricezione) due chiamate a DELARS per ogni bit di stop in più.

Per inserire il bit di parità invece, dovrete lavorare maggiormente, e per questo motivo ci lasciamo con questo compito da svolgere, dato che a questo punto del corso dovreste essere in grado di eseguire da soli queste piccole modifiche.

continua

LA PROGRAMMAZIONE IN ASSEMBLER

Continuiamo il nostro corso che ormai sta arrivando, dopo un anno dal suo inizio, alla sua conclusione: siamo infatti arrivati ad affinare le tecniche di utilizzo delle potenzialità dei Pic

Paola Sbrana - 11ª parte
su gentile concessione della Microlabs

I microcontrollori della Microchip hanno una caratteristica che non tutti possiedono: la possibilità di andare in SLEEP, ovvero di "addormentarsi" durante l'esecuzione di un qualsiasi programma, ovviamente sotto il controllo del progettista!

Questa feature fa sì che il chip assorba pochissimo, infatti si registrano correnti inferiori al microamper, permettendo di fatto un'agevole alimentazione a pile o a batteria.

Una applicazione che abbiamo già visto alcuni anni fa è la tastiera a combinazione per cassaforti: in quel caso, avevamo necessità di contenere i consumi per evitare di dover far intervenire un fabbro nel momento in cui le 4 pile stilo cessavano la loro produzione dei classici 1,5 volt.

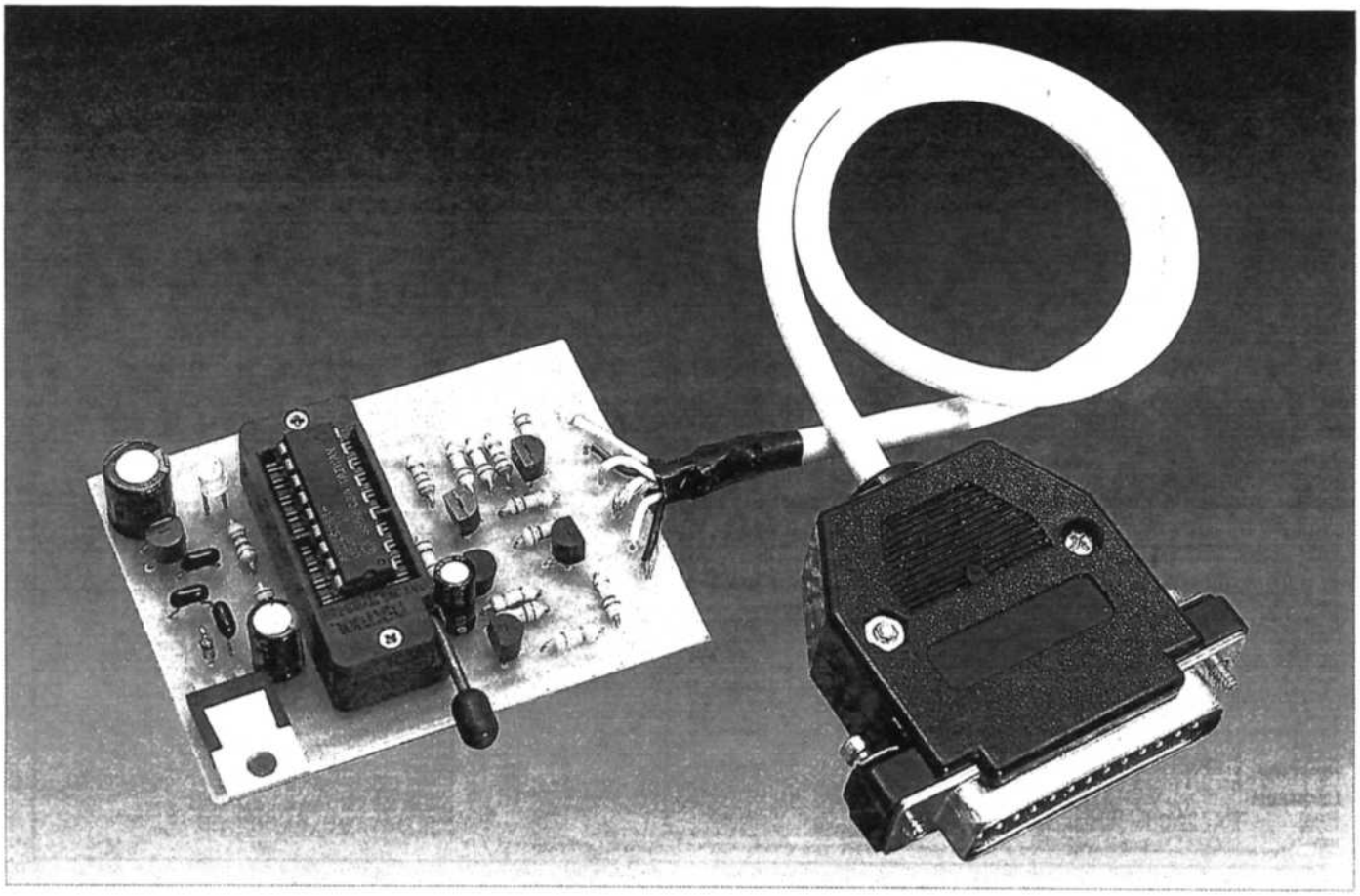
In quel circuito impiegammo un controller tipo 16C57, ovvero uno dei primi nati della famiglia 16Cxx, con la relativa circuitazione esterna per sul controllo del Master Clear.

Sulle nuove famiglie invece, non è necessario adottare lo stesso sistema, ma è sufficiente sfruttare uno degli interrupt come risveglio.

L'istruzione SLEEP

Durante la stesura del programma, l'istruzione che ci permette di far entrare il controller in modalità sleep è proprio l'istruzione chiamata "SLEEP"!

Quando si giunge a questa istruzione (che non ha operandi), il contatore del watchdog ed il suo prescaler vengono azzerati, il bit TO (Time-Out) del registro di stato viene settato a "1" ed il bit



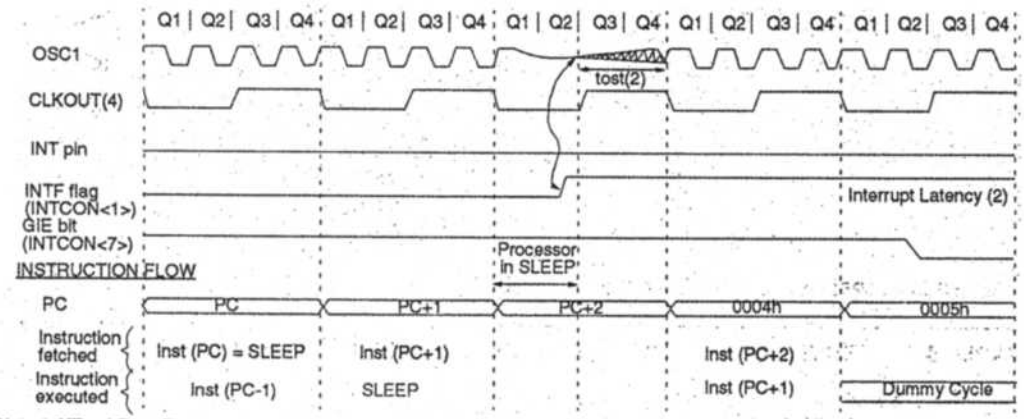


Figura 38. Timing-diagram dopo un'istruzione SLEEP

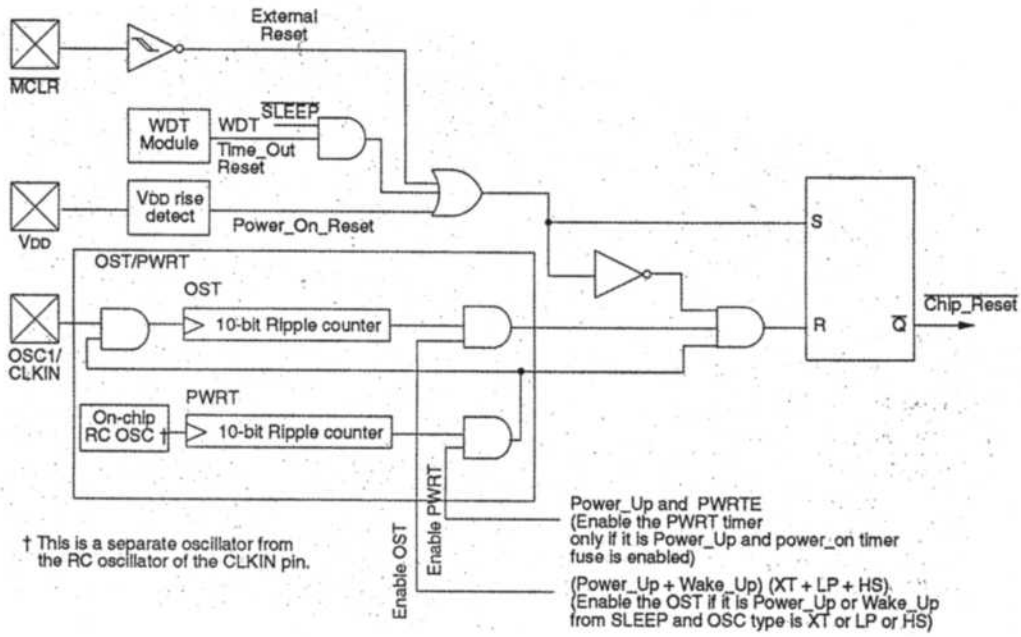


Figura 39. Diagramma a blocchi del sistema di reset del PIC16C84

PD (Power Down) dello stesso registro viene posto a "0". Inoltre, l'oscillatore viene bloccato.

Attenzione però quando parliamo di oscillatore, perché con questo termine intendiamo sempre l'oscillatore relativo al clock del micro, in quanto il micro ha al suo interno anche un oscillatore per l'watchdog e uno eventuale per il convertitore.

Quindi, se noi blocchiamo l'oscillatore del clock, non significa che li blocchiamo tutti. Per quanto riguarda l'oscillatore del watchdog comunque, per determinate applicazioni è possibile disabilitarlo senza compromettere la funzionalità del sistema e, in ogni caso, non assorbe la stessa corrente dell'oscillatore di clock.

Quando il chip si "risveglia", ovvero l'oscillatore di clock riprende a funzionare, si deve attendere un tempo di latenza pari a 1.024 per il tempo di clock prima che il program counter indirizzi l'istruzione da eseguire ed il controller la esegua. In pratica, se abbiamo un clock a 4 MHz, con l'istruzione di 1 microsecondo, avremo il risveglio dopo 1.024 x 1 = 1.024 microsecondi, cioè circa 1 millisecondo, come si può vedere nella Tabella 7 e nella Figura 38.

Dobbiamo far necessariamente notare che ben diverso è il risveglio da interrupt o da reset: infatti, nel primo caso il program counter si carica con l'indirizzo del vettore di interrupt, nel secondo caso si carica con l'indirizzo del vettore di reset. In Figura 39 è possibile vedere il diagramma a blocchi dell'intero sistema di reset del PIC16C84, mentre nella Tabella 8 si può capire che cosa cambia nei registri dopo tale operazione.

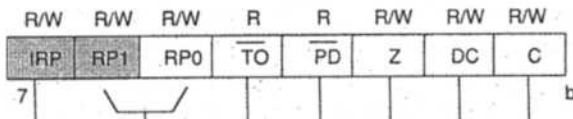
In effetti, dopo un qualsiasi reset, è possibile capire quale sia stata la causa andando ad interrogare i bit TO e PD del registro di stato. In Figura 40 riportiamo il contenuto del registro di stato, sempre utile da avere sott'occhio, mentre dalla Tabella 9 possiamo capire quale sia stata la causa del reset o del risveglio.

Il nostro software

Per valutare praticamente l'istruzione di SLEEP, abbiamo bisogno di costruirci una piccola basetta di prova, fra l'altro simile ad una già presentata alcuni mesi fa per il test della EEPROM interna e visibile in Figura 41. Il programma che proponiamo invece, è siglato PROG19 e funziona in questo modo: dopo l'accensione si mette in sleep e si risveglia solo quando uno dei quattro pulsanti viene premuto.

Tabella 7. Tempistica di risveglio del controller

Oscillator Configuration	Power-up PWRTE = 1	Power-up PWRTE = 0	Wake-up from SLEEP
XT, HS, LP	72 ms + 1024 Tosc	1024 Tosc	1024 tosc
RC	72 ms	–	–



Register: STATUS	R/W: Readable & writable
Address: 03h or 83h	R: Read only
Reset value: 000??XXX	U: Unimplemented, read as '0'
TO, PD are set or cleared specially	

C: Carry/borrow bit.
For ADDWF and ADDLW instructions.
1 = A carry-out from the most significant bit of the result occurred. Note that a subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.
0 = No carry-out from the most significant bit of the result.
Note: For Borrow the polarity is reversed.

DC: Digit carry/borrow bit.
For ADDWF and ADDLW instructions.
1 = A carry-out from the 4th low order bit of the result occurred.
0 = No carry-out from the 4th low order bit of the result.
Note: For Borrow the polarity is reversed.

Z: Zero bit.
1 = The result of an arithmetic or logic operation is zero.
0 = The result of an arithmetic or logical operation is not zero.

PD: Power down bit.
1 = After power-up or by a CLRWDT command.
0 = By execution of the SLEEP instruction.

TO: Time-out bit.
1 = After power-up and by the CLRWDT and SLEEP instruction.
0 = A watchdog timer time-out has occurred.

RP<1:0>: Register page select bits for direct addressing.

00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)

Each page is 128 bytes.
Only the RP0 bit is used by the PIC16C64. RP1 should be programmed as '0'. Using the RP1 bit as a general purpose read/write bit is not recommended, since this may affect upward compatibility with future products.

IRP: Register page select bits for indirect addressing.
0 = Bank 0,1 (00h - FFh)
1 = Bank 2,3 (100h - 1FFh)

The IRP bit is not used by the PIC16C64. IRP should be programmed as '0'. Use of the IRP bit as a general purpose read/write bit is not recommended, since this may affect upward compatibility with future products.

Figura 40. Il registro di stato

Tabella 8. Modifiche ai registri dopo un reset

	PCL Addr: 02h	STATUS Addr: 03h	PCON Addr: 8Eh
Power-On Reset	000h	0001 1xxx	---- --0-
MCLR reset during normal operation	000h	0001 1uuu	---- --u-
MCLR reset during SLEEP	000h	0001 0uuu	---- --u-
WDT reset during normal operation	000h	0001 1uuu	---- --u-
WDT during SLEEP	PC + 1	uuu0 0uuu	---- --u-
Interrupt wake-up from SLEEP	PC + 1 (1)	uuu1 0uuu	---- --u-

Legend: *u = unchanged x = unknown - = unimplemented bit, reads as '0'*
Notes: *1. When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h)*

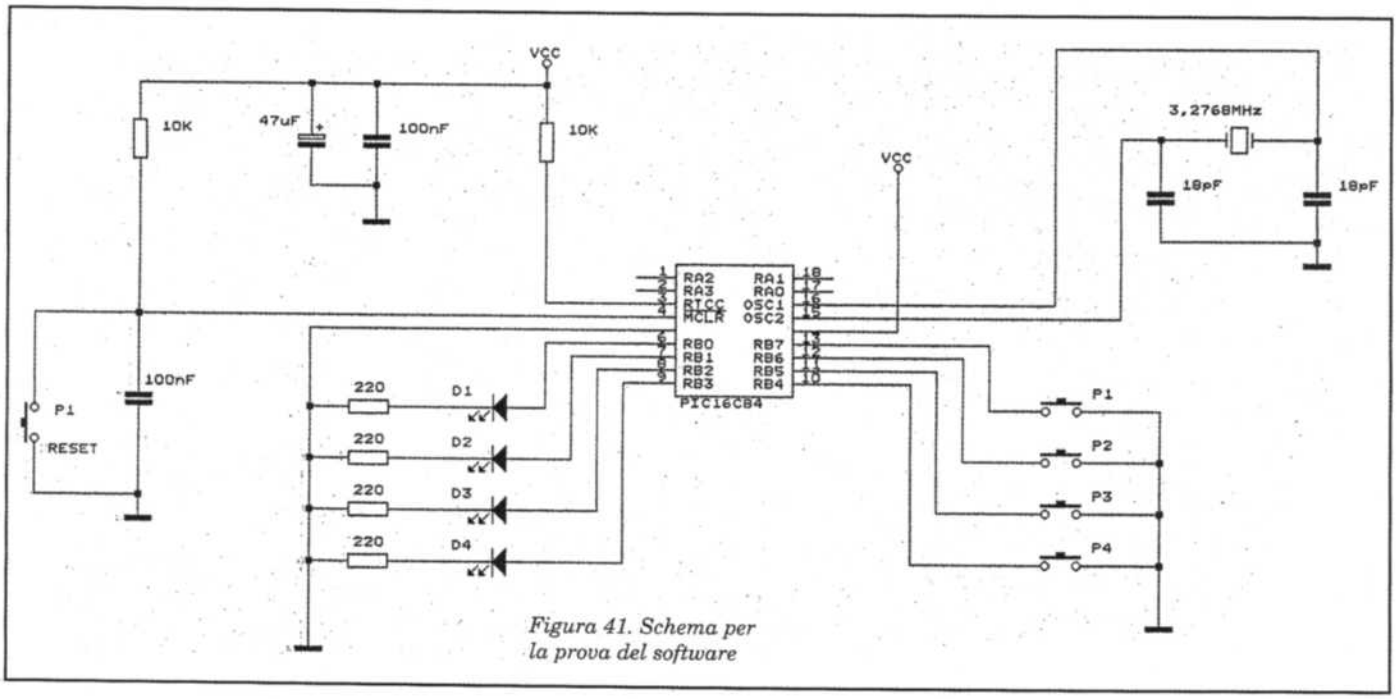


Figura 41. Schema per la prova del software

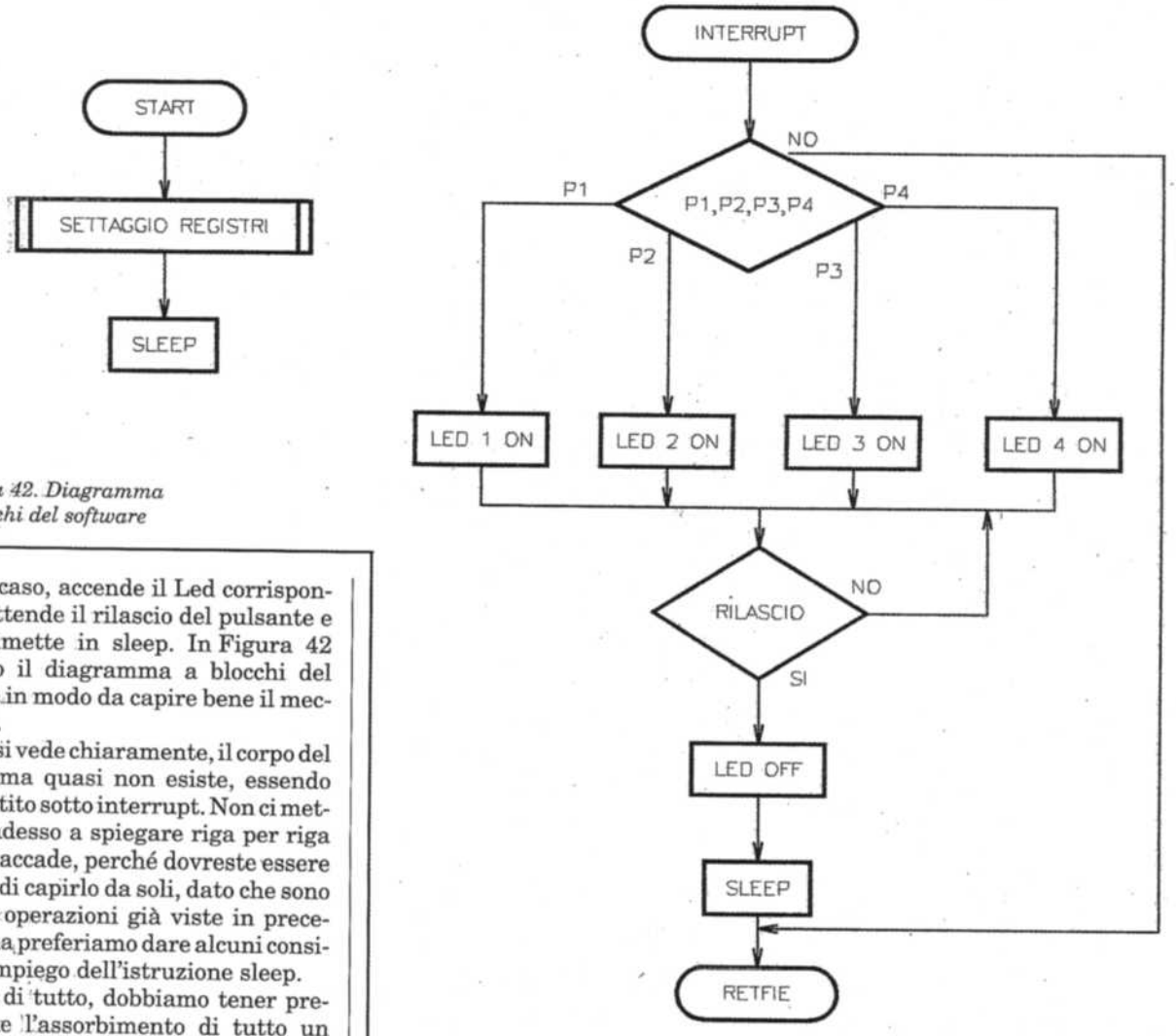


Figura 42. Diagramma a blocchi del software

In tal caso, accende il Led corrispondente, attende il rilascio del pulsante e poi si rimette in sleep. In Figura 42 troviamo il diagramma a blocchi del software in modo da capire bene il meccanismo.

Come si vede chiaramente, il corpo del programma quasi non esiste, essendo tutto gestito sotto interrupt. Non ci metteremo adesso a spiegare riga per riga che cosa accade, perché dovreste essere in grado di capirlo da soli, dato che sono eseguite operazioni già viste in precedenza, ma preferiamo dare alcuni consigli sull'impiego dell'istruzione sleep.

Prima di tutto, dobbiamo tener presente che l'assorbimento di tutto un

PROGRAMMA 19

```
TITLE 'PROG19: Prova SLEEP per Progetto'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC    EQU 01H         ; Real Time Clock Counter
PCL     EQU 02H         ; Program Counter
STAT    EQU 03H         ; Registro di stato
PORTA   EQU 05H         ; Porta A
PORTB   EQU 06H         ; Porta B
EDATA   EQU 08H         ; Data EEPROM
EADR    EQU 09H         ; Address EEPROM
ECON1   EQU 88H         ; Stato delle operaz. in EEPROM
ECON2   EQU 89H         ; Vedi ECON1
INTCON  EQU 0BH         ; Registro abilitazione interrupt
TR_A    EQU 85H         ; Tris A
TR_B    EQU 86H         ; Tris B
OPTIO   EQU 81H         ; Registro OPTION
;-----------------------------------------------------------------
FR01    EQU 0CH         ;
FR02    EQU 0DH         ;
FR03    EQU 0EH         ;
WTEMP   EQU 10H         ;
STEMP   EQU 11H         ;
;-----------------------------------------------------------------
#define P1      PORTB,7 ; Pulsante 1
#define P2      PORTB,6 ; Pulsante 2
#define P3      PORTB,5 ; Pulsante 3
#define P4      PORTB,4 ; Pulsante 4
#define RP0     STAT,5  ; Flag selezione banco ram
#define RBIF    INTCON,0 ; RB Interrupt Flag
#define RBIE    INTCON,3 ; RB Interrupt Enable
#define GIE     INTCON,7 ; Global Interrupt Enable
#define PD      STAT,3  ; Power Down bit
;-----------------------------------------------------------------
        ORG     0
        goto    START   ; Reset vector

        ORG     4       ; Interrupt vector
        movwf   WTEMP   ; potrebbe essere in banco 0 o 1
        swapf   STAT,0  ; Swap STAT in TEMP
        bcf     RP0     ; Selezione banco 0
        movwf   STEMP   ; Memorizzo stato
                        ;
        btfss   RBIF    ; Interrupt da RB?
        goto    DORMI   ; Uscita dalla subroutine di int.
        btfss   P1      ; Testo se premuto P1
        goto    PULS1   ;
        btfss   P2      ; Testo se premuto P2
        goto    PULS2   ;
        btfss   P3      ; Testo se premuto P3
        goto    PULS3   ;
        btfss   P4      ; Testo se premuto P4
        goto    PULS4   ;
        goto    DORMI   ;
PULS1   movlw   b'11110001' ; Accensione led 1
        movwf   PORTB   ;
        goto    RIL     ; Memorizza nuovo stato
PULS2   movlw   b'11110010' ; Accensione led 2
        movwf   PORTB   ;
        goto    RIL     ; Memorizza nuovo stato
PULS3   movlw   b'11110100' ; Accensione led 3
        movwf   PORTB   ;
        goto    RIL     ; Memorizza nuovo stato
PULS4   movlw   b'11111000' ; Accensione led 4
        movwf   PORTB   ;
        goto    RIL     ; Memorizza nuovo stato
RIL     call    DELA2M  ; Attesa rilascio pulsante
        btfss   P1      ;
        goto    RIL     ;
        btfss   P2      ;
        goto    RIL     ;
        btfss   P3      ;
        goto    RIL     ;
        btfss   P4      ;
        goto    RIL     ;
DORMI   movlw   b'0000' ; Setta uscite PORTA a zero
        movwf   PORTA   ;
        movlw   b'11110000' ; Setta uscite PORTB a zero
        movwf   PORTB   ;
        bcf     RBIF    ; Reset RBIF
                        ;
ENDRB   bsf     RP0     ;
        bsf     GIE     ; Abilita interrupt
        bsf     RBIE    ; Abilita RB Interrupt
        bcf     RP0     ;
        swapf   STEMP,0 ; Ripristino stato
        movwf   STAT    ;
        swapf   WTEMP,1 ; Ripristino W
        swapf   WTEMP,0 ;
        sleep           ;
;-------------------- subroutine RITARDO 20 mS --------------------
; con frequenza 3,2768 MHz
;-----------------------------------------------------------------
DELA2M  bcf     FR01,7  ; Memoria P1
        bcf     FR01,6  ; Memoria P2
        bcf     FR01,5  ; Memoria P3
        bcf     FR01,4  ; Memoria P4
LPDEL   clrwdt          ; Azzera watchdog
        nop             ; Nessuna operazione
        nop             ;    "
        nop             ;    "
        nop             ;    "
        movf    RTCC,0  ; Controllo se finiti 20mS
        SKPZ            ;
        goto    LPDEL   ;
        movlw   .128    ; Carico 128 in W
        movwf   RTCC    ; Copio W nel registro RTCC
        return          ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START   bsf     STAT,5  ; Seleziona SRAM banco 1
        movlw   b'0000' ; RA out
        movwf   TR_A    ;
        movlw   b'11110000' ; RB0..RB3 out  RB4..RB7 in
        movwf   TR_B    ;
        clrf    INTCON  ; Disabilita interrupt
        movlw   b'01000110' ; Prescaler 1:128
        movwf   OPTIO   ; Copia W in OPTION
        bcf     STAT,5  ; Seleziona SRAM banco 0
        movlw   .128    ; Settaggio iniziale RTCC
        movwf   RTCC    ;
        movlw   b'0000' ; Setta a zero uscite su PORTA
        movwf   PORTA   ;
;-------------------- main program --------------------
MAIN    call    DELA2M  ; Attesa 20mS
        goto    DORMI   ;
;-----------------------------------------------------------------
    END
```

Tabella 9. Effetti di un reset sui bit TO e PD del registro di stato

Event	TO	PD	Remarks
Power-up	1	1	–
WDT Time-out	0	U	No effect on PD
SLEEP instruction	1	0	–
CLRWDT instruction	1	1	–

U: unchanged
Note: *A WDT timeout will occur regardless of the status of the TO bit. A SLEEP instruction will be executed, regardless of the status of the PD bit. Table 3.9.2.2 reflects the status of PD and TO after the corresponding event.*

TO	PD	RESET was caused by
0	0	WDT wake-up from SLEEP
0	1	WDT time-out (not during SLEEP)
U	0	$\overline{MCLR}$ wake-up from SLEEP
1	1	Power-up
U	U	$\overline{MCLR}$ reset during normal operation

U: unchanged
Note: *The PD and TO bit maintain their status until an event of Table 3.9.2.1 occurs. A low-pulse on the $\overline{MCLR}$ input does not change the PD and TO status bits.*

circuito, non è dato solamente dall'assorbimento del chip, ma dalla totalità degli assorbimenti. Quindi, se per esempio una volta in sleep facciamo rimanere un led acceso, l'assorbimento totale sarà dato proprio dall'assorbimento di tale led. Per questo motivo, nel nostro esempio abbiamo deliberatamente lasciato le porte non utilizzate (porta A) e quelle relative dei led a zero, ma ciò non basta, perché così facendo una piccola corrente continuerà a scorrere su quei pin, basterà che lo proviate con un multimetro.

Allora dovrete, prima di inserire l'istruzione sleep, portare in ingresso (che equivale ad un three-state) tutti i pin con la modifica dei registri TRISX, dove X indica la porta da settare. Quando poi il chip si risveglierà, dovrete rimpostare le direzioni delle porte, come dopo ogni reset. Noterete che così facendo otterrete assorbimenti inferiori al microamper (semprechè abbiate disabilitato l'watchdog).

continua

LA PROGRAMMAZIONE IN ASSEMBLER

È passato un anno intero da quando abbiamo incominciato il nostro corso di programmazione dei microprocessori della famiglia Pic, i chip in tecnologia Risc che hanno riscosso un incredibile successo. In quest'ultima puntata affiniamo gli ultimi dettagli, rimandando i lettori al mese prossimo per le applicazioni vere e proprie

Paola Sbrana - 12ª parte
su gentile concessione della Microlabs

Concludiamo in queste pagine il corso di programmazione in assembler dedicato ai controller della Microchip, gettando uno sguardo ai sistemi di sviluppo, alla documentazione ed ai programmatori ufficiali e non.

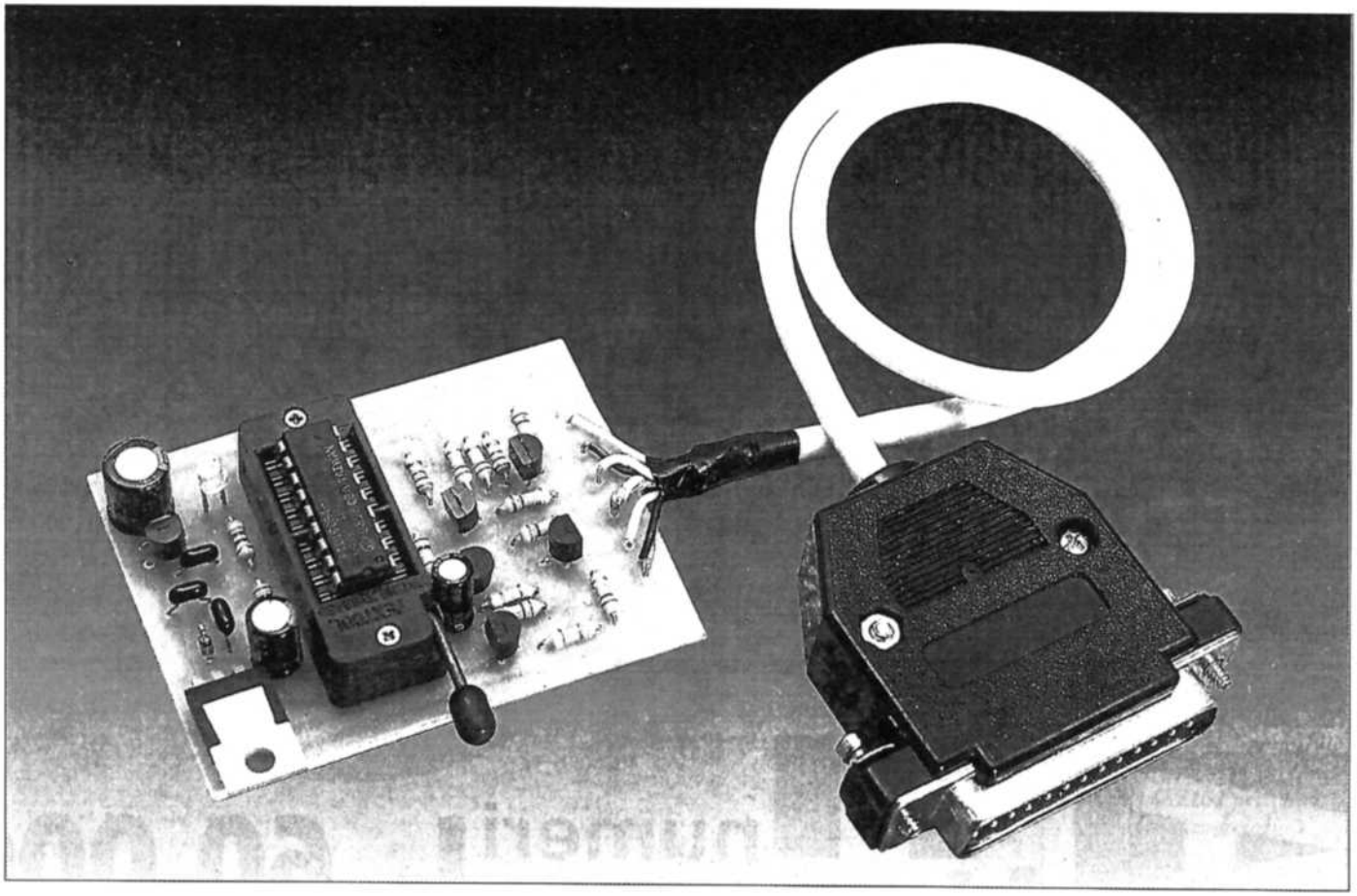
Fino ad adesso, lo scopo del corso è stato quello di offrire agli hobbisti un facile accesso al mondo dei microcontroller, dettagliando ampiamente ogni passo per consentire a tutti di entrare in possesso delle capacità di gestione di un microcontroller, dato che la documentazione ufficiale è rigorosamente in lingua inglese che non tutti la comprendono perfettamente.

L'unico neo della documentazione offerta dalla Microchip, infatti, è che non esiste nemmeno una pagina in lingua italiana, eccetto quella in cui vengono elencati i distributori nazionali.

Ma non dobbiamo preoccuparci più di tanto, perché altre case produttrici di microcontroller distribuiscono la documentazione dei loro prodotti in quantità nettamente inferiore alla Microchip.

Quindi, l'unico approccio possibile, fino alla pubblicazione del nostro corso, era quello in lingua inglese.

Da questo mese inoltre, sarà disponibile anche un manuale di programmazione che riprenderà tutti gli articoli pubblicati fino ad oggi ed in più ne



conterrà altri che consentiranno di dialogare con diverse periferiche e di costruirsi delle routine che poi potranno essere impiegate ogni volta che sarà necessario senza più doversi ricordare il perché di una determinata operazione o istruzione.

Nuove periferiche

Vediamo allora quali sono le periferiche che riusciremo a gestire con le routine descritte nel manuale.

Nel capitolo 12 viene trattato l'argomento dei display alfanumerici a N righe e M colonne (l'esempio è orientato su un 16 colonne e 2 righe), mostrando le routine per scriverci sopra sia con protocollo a 4 che ad 8 bit.

Nel capitolo tredici vengono, invece, lette e scritte le memorie tipo 93xx, ovvero quelle che dialogano con il bus Microwire.

Sarà possibile scrivere in una cella un valore e poi rileggerlo.

Nel capitolo quattordici, invece, faremo la stessa cosa ma con le memorie di tipo 24xx, ovvero quelle che dialogano con il bus IIC.

Sia con le prime che con le seconde routine, sarà possibile gestire interamente le EEPROM seriali viste, leggendole, scrivendoci sopra e cancellandole a piacere.

Nel capitolo successivo troverete le routine che servono per interfacciarsi con le chiavi DALLAS con il protocollo detto "one-wire", ovvero a filo unico.

Queste routine permetteranno di leggere il codice di una chiave DALLAS, implementando anche il riconoscimento del CRC.

Nel sedicesimo capitolo, viene spiegato come collegare al microcontroller uno o più shift-register, in modo tale da moltiplicare il numero delle uscite.

Nel capitolo diciassette, vengono date delle routine per la generazione di un segnale PWM preciso con variazione del duty-cycle, per l'impiego con carichi in corrente continua.

Nell'ultimo capitolo infine, vengono descritte quelle modifiche da apportare ai programmi ed alle routine per passare da un chip della famiglia base (PIC16C5x) alle altre e viceversa.

Saranno poi presenti delle appendici con le istruzioni speciali e le direttive dell'assemblatore.

Unitamente al manuale, viene fornito un dischetto in cui saranno memorizzati, oltre ai programmi descritti nel manuale, anche l'assemblatore ufficiale, un disassemblatore e un eseguibile che consentirà al computer di divenire un terminale per lo scambio di dati sulla linea seriale, con la modifica di tutti i parametri interessanti come la velocità, il numero di bit di dato e di stop, la rappresentazione dei byte (carattere, binario o byte) e la parità.

Il corso completo è stato raggruppato in una raccolta integrata da un dischetto

Il programmatore necessario

Come per qualsiasi altro componente programmabile, il PIC deve essere scritto con dei parametri ben precisi.

Se ricordate, nella prima puntata del corso apparsa su Progetto Elektor di Gennaio '96, abbiamo presentato un piccolo circuito da applicare sulla porta parallela del computer e da pilotare con un software da richiedere alla Microlabs e che era in grado di programmare soltanto i PIC16C84.

Ovviamente, tale programmatore era di tipo sperimentale, poiché, tanto per fare un esempio, gli stessi PIC16C84 in versione a montaggio superficiale non venivano scritti correttamente. Per ovviare a ciò, la Microchip stessa ha commercializzato ultimamente una nuova serie di prodotti per la programmazione dei PIC compatibili con le norme CE.

Nuovi programmatori

In Figura 43 vediamo l'ultimo nato: il Picstart Plus. Questo programmatore viene venduto ad un prezzo accessibile a tutti e riesce a programmare tutti i chip Microchip presenti e futuri, essendo upgradable. La caratteristica principale è proprio la possibilità di programmare tutti i componenti Microchip, compresi i nuovissimi:

- PIC12C508,
- PIC12C509,

nonché tutti i:

- PIC16C5x,
- PIC16Cxx,
- PIC17Cxx e
- PIC14000, ancora poco conosciuto in Italia.

Il software di gestione è completamente nuovo e gira sotto Windows 3.1 o Windows 95.

Consente di lavorare in un ambiente integrato detto MPLAB e di non uscirne fino al termine della progettazione (simulatore e assemblatore integrati). Il collegamento con il computer avviene tramite linea seriale RS232.

In Figura 44 potete vedere una delle possibili configurazioni dell'ambiente di sviluppo MPLAB con il simulatore incorporato.

Il Picstart Plus però non viene consigliato per le grandi produzioni, ovvero per quelle ditte che programmano migliaia di PIC all'anno oppure che necessitano di programmare componenti in smd: in questo caso si deve impiegare il Promate II, il programmatore visibile in Figura 45.

Anche questo si collega al computer tramite la porta seriale e lavora sotto Windows o sotto dos, ma non ha uno zoccolo unico: lo zoccolo per la programmazione dei PIC varia in funzione del PIC da programmare.

In realtà, se si devono programmare PIC in versione a montaggio superficiale, questo programmatore è l'unico che dia dei risultati ottimi, con un errore prossimo allo 0%. Purtroppo il costo di questo programmatore non è così basso come quello del Picstart Plus ed inoltre viene venduto con un solo zoccolo (per una determinata famiglia).

Se avete da programmare chip di diversa famiglia (PIC16C5X, PIC16C6x, Pic16C7x, PIC17Cxx) o in versione SMD, dovete acquistare lo zoccolo appropriato con un costo che si aggira intorno alle 200-300.000 lire.

Questo programmatore, inoltre, consente la programmazione stand-alone, ovvero memorizza sulla sua EEPROM interna il programma da inserire nei PIC e poi può programmare i vari PIC anche in assenza del computer. Se ad esempio volete fare delle copie di chip non protetti, sarà sufficiente leggerli (aiutandosi con le informazioni che appaiono sul display alfanumerico retroilluminato) e poi programmarli senza nemmeno accendere il computer.

Per informazioni sui programmatori della Microchip, potete contattare il nostro consulente allo 0347/2643514.

I sistemi di sviluppo

Di sistemi di sviluppo per microcontrollori ne abbiamo già parlato nel mese di settembre '96, e quindi adesso ricordiamo che in commercio troviamo sia prodotti Microchip sia prodotti forniti da terze parti ma approvati egualmente dalla Microchip. Ovviamente, i prodotti ufficiali offrono una garanzia in più rispetto agli altri, e cioè la non interruzione di assistenza tecnica in caso di guasto, cosa che potrebbe accadere per importazioni parallele.

Attualmente comunque, sia i prodotti diretti che gli altri non hanno causato problemi agli acquirenti, segno che la qualità è abbastanza elevata.

Il sistema di sviluppo offerto da Microchip è quello visibile in Figura 46 e detto PICMASTER. Ovviamente, anche per questo prodotto vale lo stesso discorso fatto per il programmatore, e cioè che per ogni famiglia di chip è necessario un probe diverso, e questa volta i vari probe costano mediamente sulle 800.000 lire cadauno.

Il vantaggio però di poter lavorare in realtime con break-point, watchdog e la visualizzazione dei registri interni con possibilità di modifica è palese: si riesce a debuggare un programma in circa un quinto di tempo che non con i chip finestrati ed il simulatore.

Ovviamente, noi consigliamo l'acquisto di questo prodotto solo nei casi in cui venga sfruttato per almeno 6 ore al giorno, oppure in occasione di un lavoro che ne consenta un rapido ammortamento.

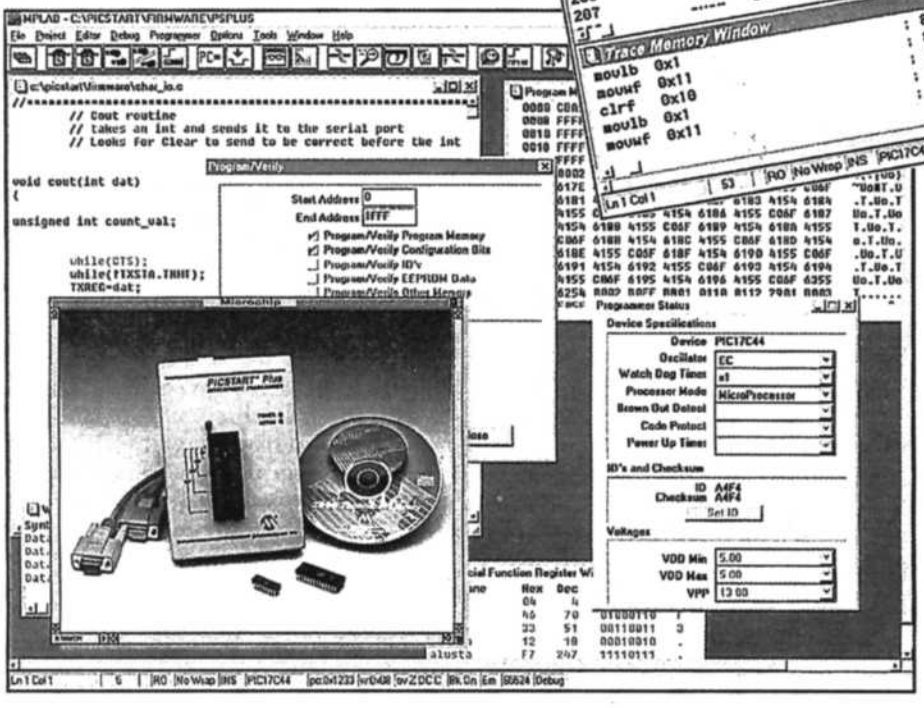


Figura 43. Il programmatore Picstart Plus

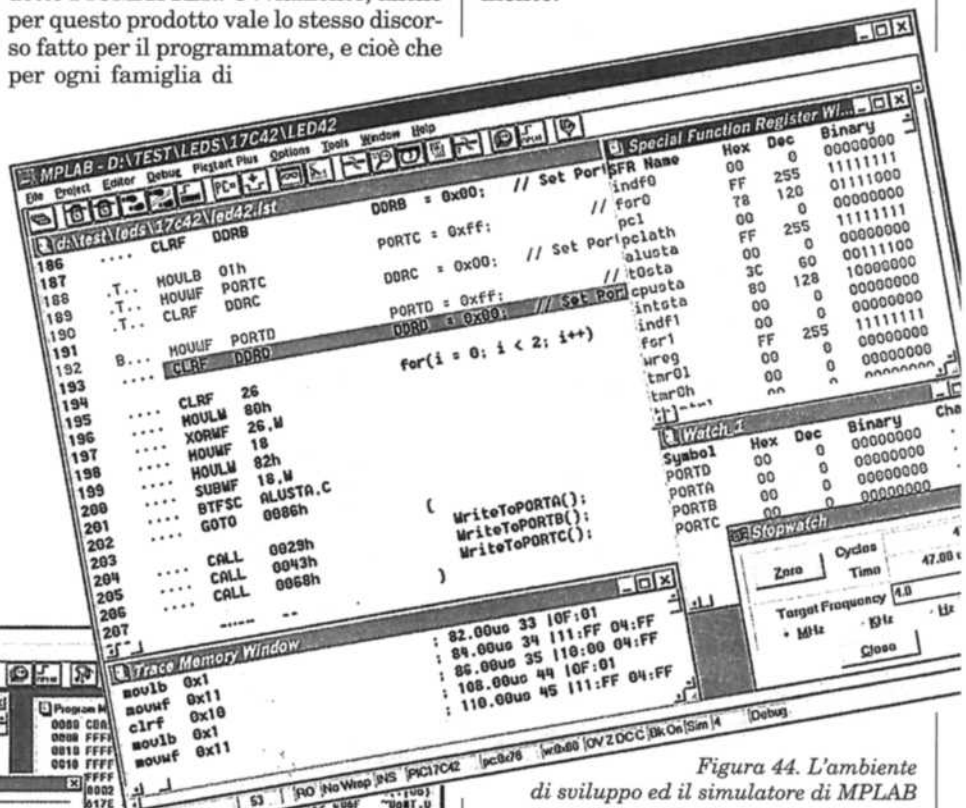


Figura 44. L'ambiente di sviluppo ed il simulatore di MPLAB

Per quanto riguarda il software disponibile, oltre all'MPLAB visto precedentemente, è da poco disponibile anche il compilatore "C".

Non l'abbiamo ancora provato, ma da esperienze precedenti continuiamo a credere che per determinate applicazioni il linguaggio "C" sia inappropriato. Una novità, invece, è costituita dal software MP-DriveWay detto Application Code Generator, ovvero un generatore automatico di codice "C" da passare poi al compilatore per ottenere l'assembler.

Questo software consente, ad esempio, di gestire le porte, i registri interni, la RAM del microcontroller, semplicemente "cliccando" su pulsanti dedicati.

In pratica, con poche operazioni è possibile settare il funzionamento della periferica USART sul PIC16C73 come full-duplex, 8 bit, interrupt driven con sole tre "cliccate".

Allo stesso modo è possibile creare delle periferiche virtuali (come ad esempio la seriale asincrona) in chip in cui tali periferiche sono assenti. Nel caso precedente, è possibile far eseguire al software la gestione di una USART anche in integrati come il PIC16C84 dove tale periferica non esiste (un po come abbiamo fatto noi con le routine per il dialogo seriale).

Il vantaggio è che è possibile scrivere programmi senza conoscere né il linguaggio "C" né il linguaggio assembler.

Ma non finisce qui

Con la conclusione del nostro corso, non termina certamente l'attenzione di Progetto Elektor verso i microcontrollori della famiglia Pic, che per primi in Italia abbiamo scoperto e presentato al grande pubblico. Già dal mese prossimo, infatti, troverete nuove applicazioni realizzate nell'ottica di illustrare le modalità operative che ci hanno permesso di realizzare quel determinato progetto.

Inoltre, vi invitiamo sin dora a inviare lettere e fax in redazione per richiederci progetti particolari che dal vostro punto di vista possono essere utili a molti: le migliori idee verranno sviluppate nei nostri laboratori e pubblicate in queste pagine. Quanti, invece, hanno deciso di mettere in pratica il nostro corso e hanno realizzato progetti in piena autonomia possono inviarceli in redazione con una dettagliata descrizione funzionale, schemi elettrici, diagrammi a blocchi e tutto quanto ritenete utile per illustrare la logica operativa del vostro prodotto.

Le migliori pubblicazioni troveranno posto nelle nostre pagine.

L'indirizzo a cui scrivere e inviare le vostre collaborazioni è:

Gruppo Editoriale JCE
Redazione di Progetto Elektor
Via Ferri, 6
20092 Cinisello Balsamo ■

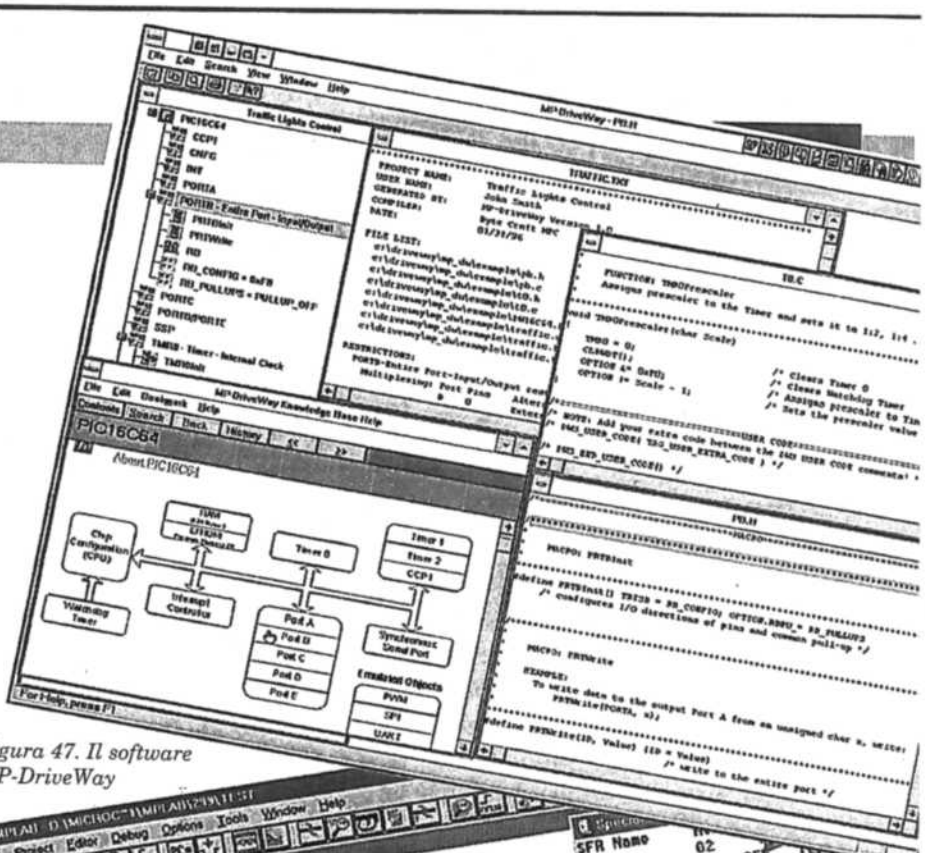


Figura 47. Il software MP-DriveWay

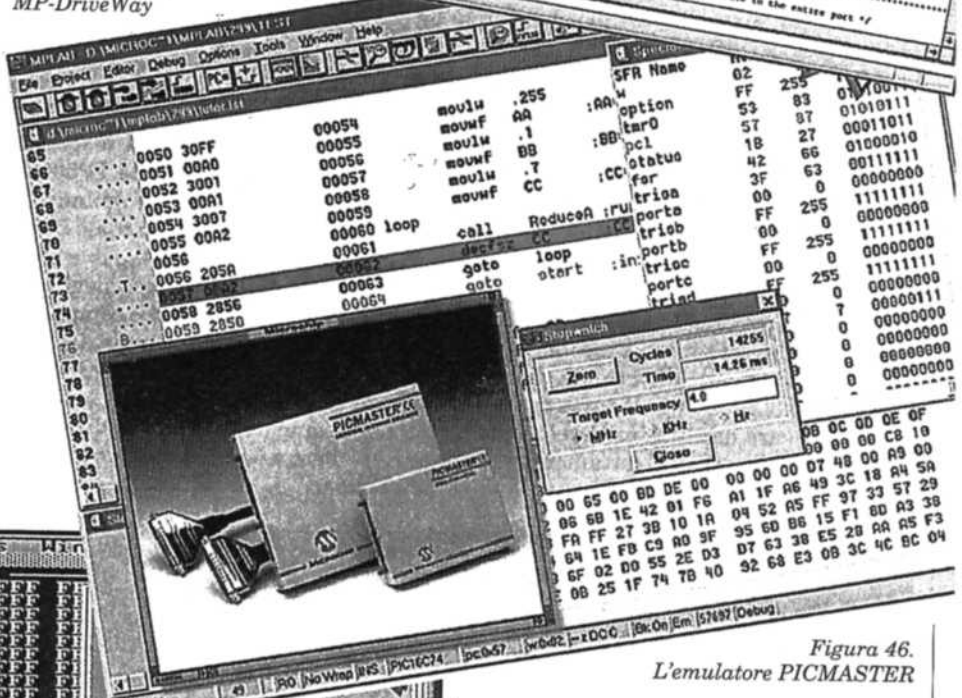


Figura 46. L'emulatore PICMASTER

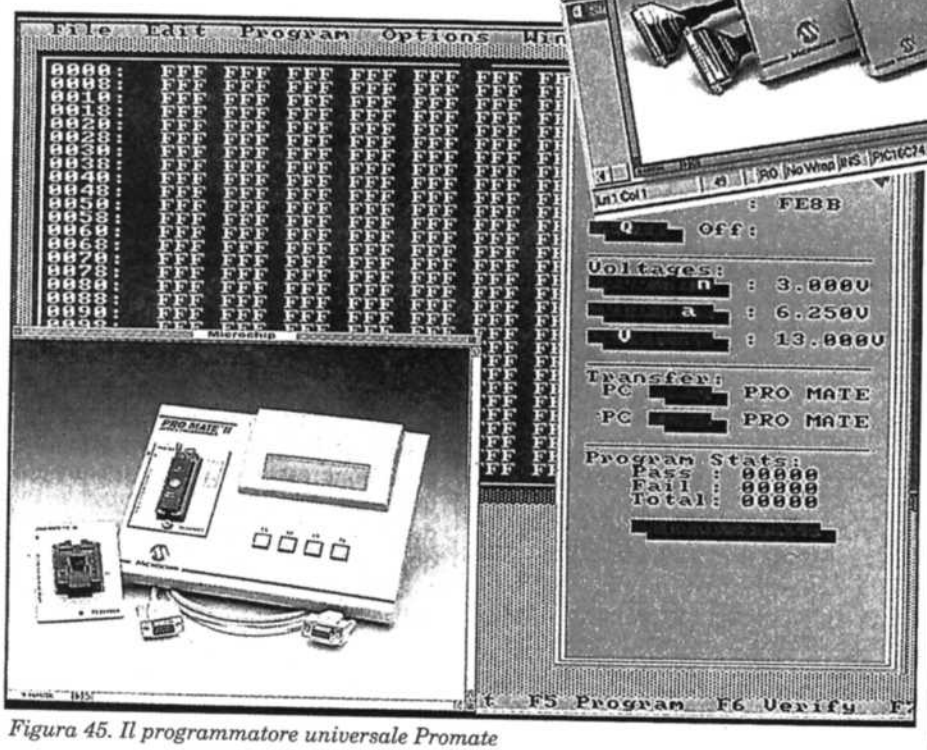


Figura 45. Il programmatore universale Promate