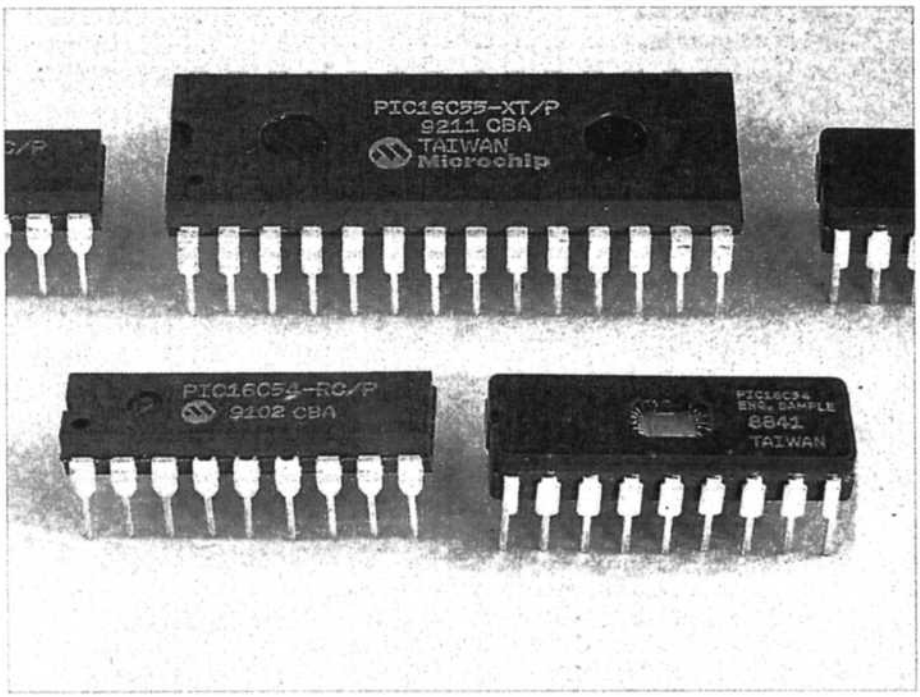
# PIC 16C5x: IMPARIAMO A PROGRAMMARLO

***Visto il grande interesse suscitato dai progetti realizzati con il controller PIC16C5x, abbiamo deciso di dedicare una serie di puntate alla sua programmazione; una volta avviati a un buon livello di preparazione, inoltre, presenteremo il progetto di un programmatore di PIC.***

*di Andrea Sbrana IW5CBO - 1ª parte*




**Caratteristiche tecniche**

Set di istruzioni limitato (33)
Massima velocità operativa 20 MHz
Eprom interna da 512 byte a 2 K x12 bit
SRAM interna costituita da 25 a 72 registri a 8 bit
7 registri dedicati
2 livelli di stack
Da 12 a 20 porte bidirezionali programmabili
Prescaler interno a 8 bit
Oscillatore interno
Watchdog
Eprom mascherabile alla lettura esterna
SLEEP per risparmio di corrente
Range di funzionamento 3-5,5 V
Consumo <2 mA a 5 V 4 MHz
15 µA a 3 V 32 kHz
<3 µA in stand-by

Nel corso degli ultimi mesi, abbiamo pubblicato diversi progetti che si basavano su un nuovo tipo di controller: il PIC.
Siamo fortemente convinti sia della validità di questi componenti che della futura diffusione e per questo motivo abbiamo deciso di incominciare questo corso di programmazione.
Una volta arrivati a comprendere tutte le funzioni, inoltre, pubblicheremo un semplice progetto di programmatore di PIC che vi permetterà di mettere in pratica quanto "studiato".
Iniziamo con l'analisi delle caratteristiche peculiari di questi controller.

## PIC 16C5x: le caratteristiche

Possiamo notare che hanno un set di istruzioni molto ridotto che, se da un lato limita l'operatività, dall'altro facilita la velocità esecutiva. La EPROM interna può variare da 512 byte a 2 K x12 bit, a seconda del modello di PIC scelto.

Chi ha già un minimo di pratica sui controller, sa che 512 byte di EPROM nella maggior parte dei casi non vengono mai completamente riempiti.

Esiste, inoltre, la possibilità di mascherare il programma memorizzato per non farlo poi rileggere in alcun modo da eventuali "copiatori" di professione!

La RAM interna, a differenza di quanto ci si possa aspettare, è formata da un certo numero di registri a otto bit, indirizzabili sia direttamente che indirettamente.

Sono a disposizione anche altri sette registri dedicati, il cui utilizzo verrà analizzato in seguito.

I livelli di stack, cioè il numero di chiamate a subroutine annidate, è fissato in due, secondo il tipo di device utilizzato.

Sempre a seconda del tipo scelto, le porte di ingresso/uscita possono essere 12 oppure 20, tutte programmabili via software e con continuità: anche durante il normale funzionamento, per ogni singolo bit sia in input che output.

Tutti i tipi di PIC possiedono sia il watchdog che un prescaler: il primo, letteralmente tradotto come "cane da guardia", controlla che la CPU (termine con cui chiameremo alcune volte il PIC) non si sia in qualche modo "piantata" e, eventualmente, forza con un reset interno la riesecuzione del programma, mentre il secondo serve per

dividere o contare il clock interno oppure un segnale digitale che giunge dall'esterno.

Ci sono quattro diversi tipi di oscillatore previsti per i PIC, anche se poi in commercio ne troviamo essenzialmente due: quello quarzato, che offre le migliori prestazioni e quello a rete RC.

In Figura 1 troviamo la piedinatura dei chip PIC16C54,55,56 e 57.

## Applicazioni dei microcontroller

Prima di passare alla descrizione implementativa del chip, vogliamo soffermarci su quelle che sono le reali applicazioni di un controller di questo tipo.

Non possiamo, infatti, dichiarare che con il PIC si possa fare tutto quello che si gestisce con altri microprocessori come quelli della serie 80xx e nemmeno con personal computer.

Il PIC si inseriscono nella fascia dei microcontroller che, per la loro versatilità, per il loro ottimo rapporto prezzo/prestazioni e per la loro semplicità di programmazione consentono di pilotare dodici o venti linee digitali come, ad esempio, un display alfanumerico simile a quello utilizzato nel progetto del visualizzatore per campanelli presentato nel novembre '92, oppure un altro integrato, come l'UM91260A impiegato nel combinatore telefonico automatico presentato in febbraio '93, oppure senza altri componenti possono gestire tastiere e relè

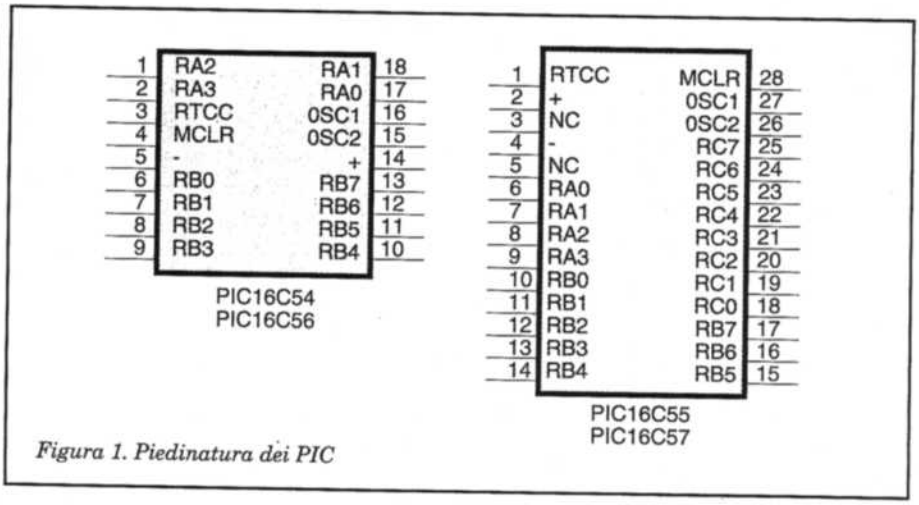



*Figura 1. Piedinatura dei PIC*

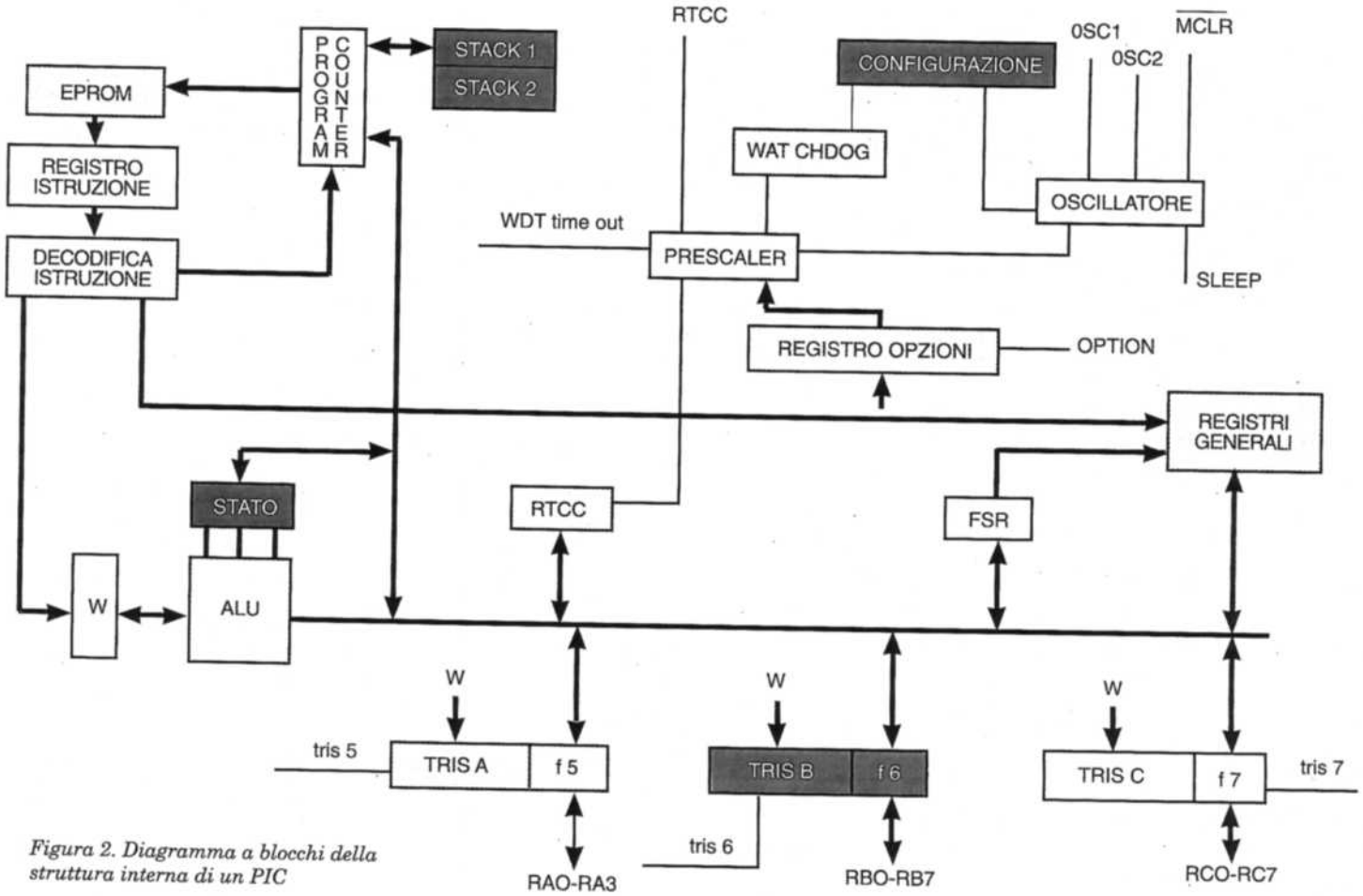



*Figura 2. Diagramma a blocchi della struttura interna di un PIC*

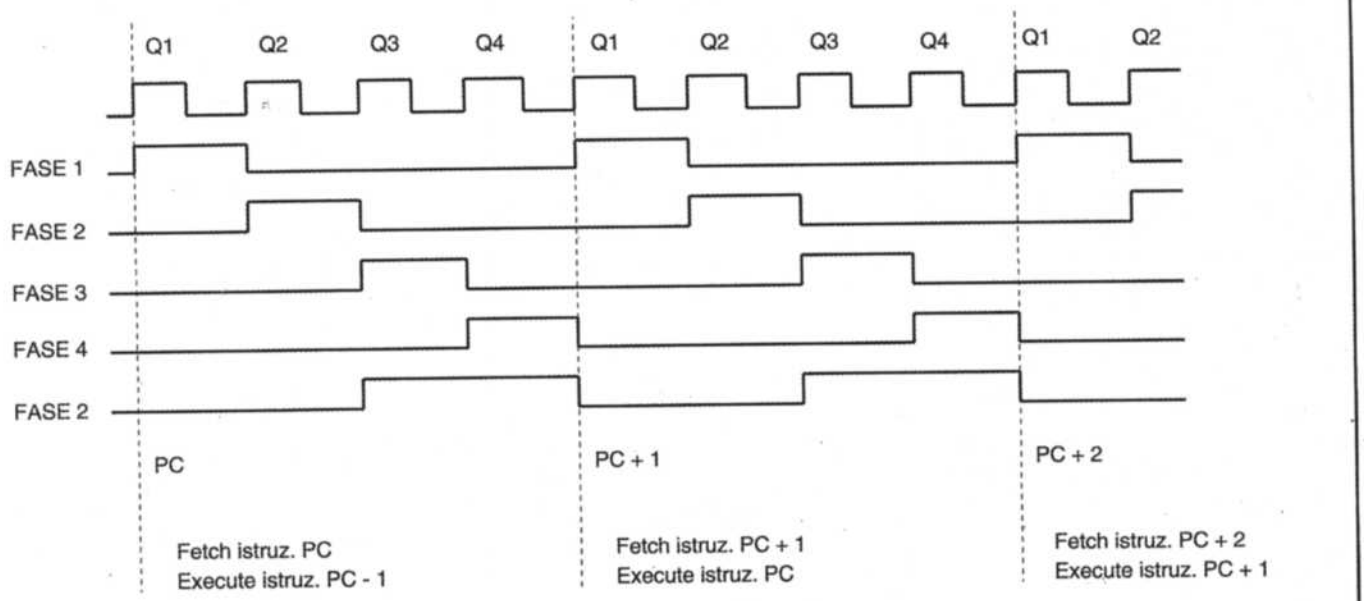



*Figura 3. Diagramma temporale del ciclo di fetch*

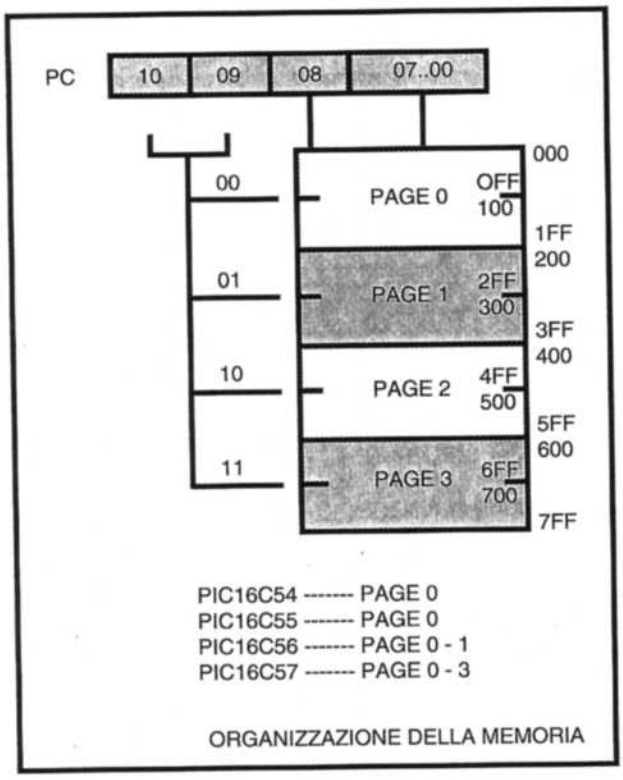



*Figura 4. Organizzazione della memoria SRAM*

come visto nel progetto della tastiera per antifurti pubblicata nel mese di gennaio '93. Ovviamente, sono possibili molte altre applicazioni, limitate solo dalla struttura interna del chip, dalla fantasia del progettista e dalle reali esigenze tecniche. Posiamo solamente citare alcune delle applicazioni più comuni in cui abbiamo utilizzato un PIC: trasmissione di dati e/o codici DTMF, controllo velocità motori, convertitori per seriale RS232, antifurti, controllo impianto elettrico di edifici, contatori, visualizzatori, telecomandi intelligenti.

## Descrizione della struttura interna

In Figura 1 vediamo la piedinatura corrispondente ai vari device attualmente in commercio, mentre in Figura 2 troviamo lo schema a blocchi.

La prima componente da vedere è la EPROM: in tale memoria il PIC immagazzina il programma che noi abbiamo scritto.

Questa memoria potrà essere di 512 byte, di 1 K oppure di 2 K, a seconda della device scelto. Ogni cella del EPROM è di 12 bit.

L'indirizzamento della istruzione viene garantito da un registro detto PROGRAM COUNTER (PC) che ha il compito di indirizzare una cella della EPROM

e di tenere conto nel far ciò di alcuni elementi fondamentali.

Il primo di questi è la cella da puntare dopo un reset (ad esempio quando viene data l'alimentazione): inizialmente il PC punta alla cella di indirizzo più alto, quindi nel caso del PIC16C54 punta alla 1FFh, dove per h intendiamo il numero espresso in esadecimale.

Altro elemento fondamentale è il livello di stack in cui il PIC si trova: in pratica ci sono due registri, STACK1 e STACK2, che memorizzano un indirizzo in base a ogni istruzione CALL (che vedremo in seguito) e con la sequenza di una memoria LIFO (Last In First Out) o, per meglio dire, di una "pila".

Se, quindi, nel programma il PIC trova una istruzione CALL, inserisce in STACK1 l'indirizzo attuale e nel PC viene memorizzato l'indirizzo a cui saltare. Se successivamente viene trovata un'altra istruzione CALL, il PIC memorizza il valore registrato in STACK1 dentro STACK2 e in STACK1 inserisce il nuovo valore attuale.

Viceversa, a ogni istruzione RETLW, viene riportato nel PC il valore presente nello STACK1 e, dopo, in STACK1 viene trasferito il valore di STACK2.

Ultimo elemento da tener presente è il "ciclo di fetch". Ogni CPU, infatti, ha il seguente modo di procedere nel suo lavoro: il PC punta a una cella di memoria che contiene l'istruzione da eseguire successivamente, istruzione che viene immagazzinata nel REGISTRO ISTRUZIONE. Da qui passa alla zona attiva di DECODIFICA ISTRUZIONE e viene eseguita, incrementando così di nuovo il PC.

In pratica, il ciclo di fetch consiste nel far puntare al PC la cella dell'istruzione successiva a quella corrispondente all'attuale operazione in svolgimento in modo tale da risparmiare tempo.

In Figura 3, potete vedere il diagramma degli stati di questi passaggi.

Infine, dato che i vari tipi di PIC hanno memoria di diversa lunghezza, anche i bit del PC variano di conseguenza, come

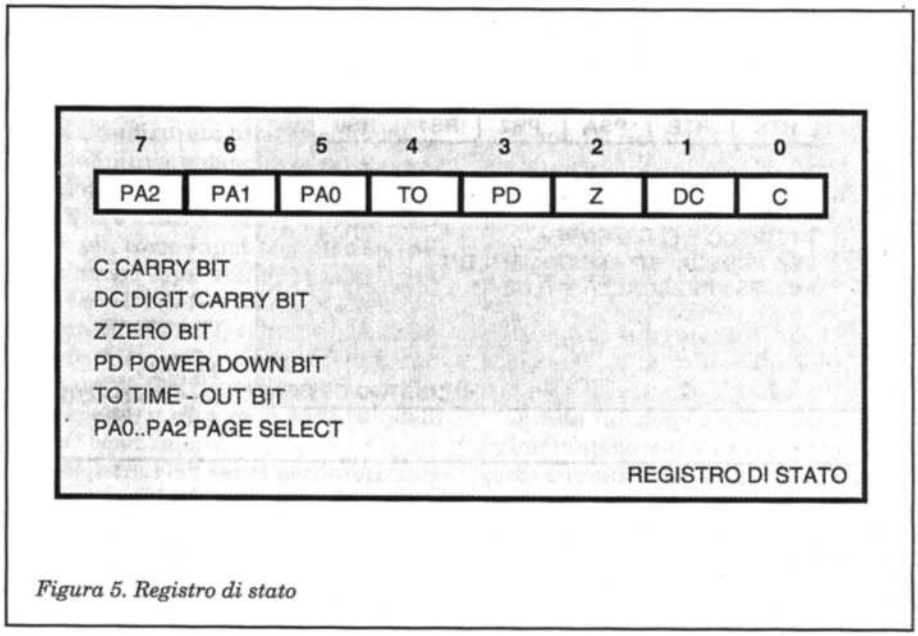



*Figura 5. Registro di stato*

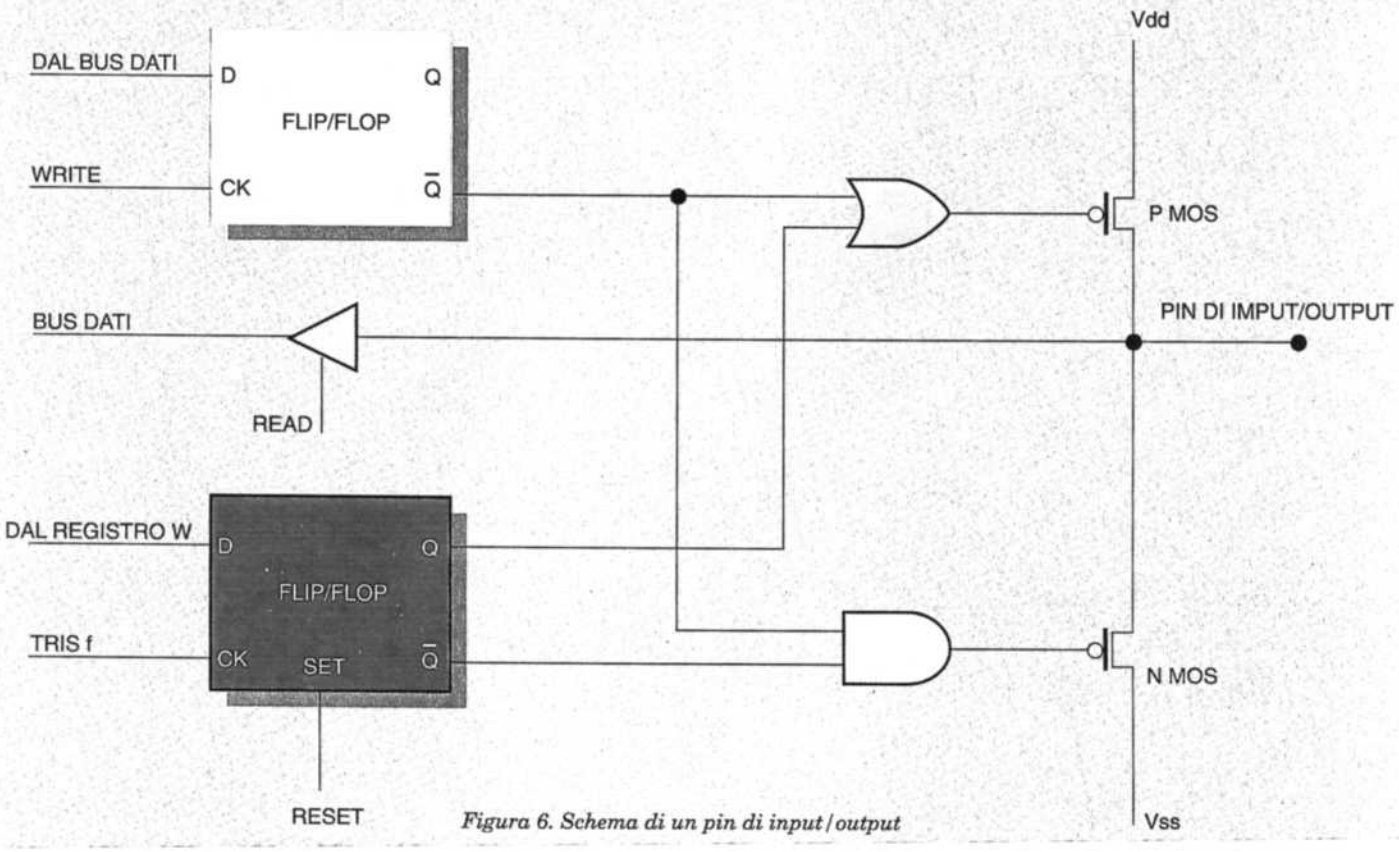



*Figura 6. Schema di un pin di input/output*

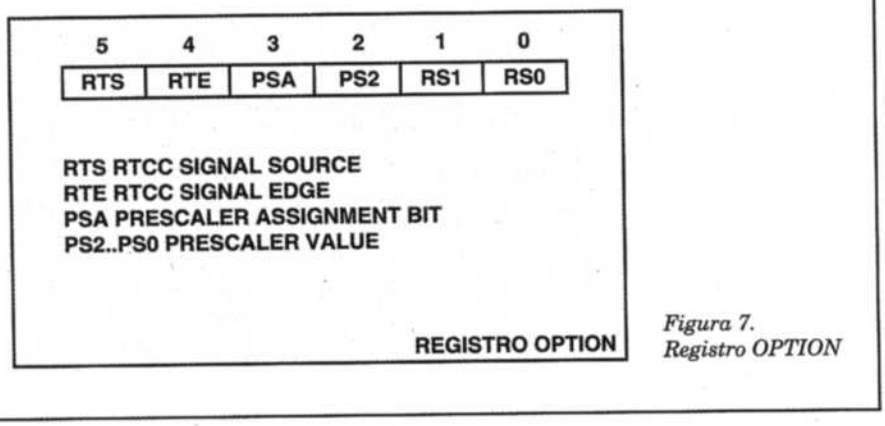



*Figura 7. Registro OPTION*

è possibile vedere nella mappa della memoria di Figura 4: per indirizzare tutti e quattro i banchi del PIC16C57 sono necessari due bit in più rispetto agli altri device. Tornando alla Figura 2, possiamo notare che dopo la decodifica è possibile lavorare con i registri generali oppure con il registro W (Word). Quest'ultimo rappresenta uno dei registri più frequentemente usati dal programmatore poiché ogni "valore" deve passare da lì.Ad esempio, per fare una somma o una divisione attraverso la ALU (Aritmetic Logic Unit) è necessario passare per W.

La ALU è la componente che esegue le operazioni logiche.

Lo stato successivo a ogni operazione del PIC viene memorizzato nel registro di STATO, visibile in Figura 5.

I tre bit PA0-PA2, servono per selezionare i banchi possibili dei registri dei vari PIC (lo vedremo in seguito); il bit di time-out (TO) viene settato a 1 durante il reset di alimentazione e dalle istruzioni CLRWDT e SLEEP, mentre viene resettato da un time-out del registro WATCHDOG.

Il bit di POWER DOWN (PD), invece, è settato a 1 dalla sola istruzione CLRWDT e resettato dall'istruzione SLEEP. Il bit ZERO Z, è settato a 1 solo se il risultato dell'ultima operazione eseguita è zero. I due bit DIGIT CARRY (DC) e CARRY (C) sono, invece, settati se sopravviene un overflow. Sempre in Figura 2, alla ALU sono connessi i registri F5, F6, F7 e i rispettivi TRIS_A, TRIS_B e TRIS_C.

Questi registri controllano lo stato delle porte di ingresso/uscita e in Figura 6 possiamo vedere come sono strutturati i pin del PIC.

Il flip-flop in alto corrisponde a un bit del registro relativo a una porta (ad esempio la A), mentre quello in basso corrisponde a un bit del registro TRIS_A (sempre in riferimento alla porta A).

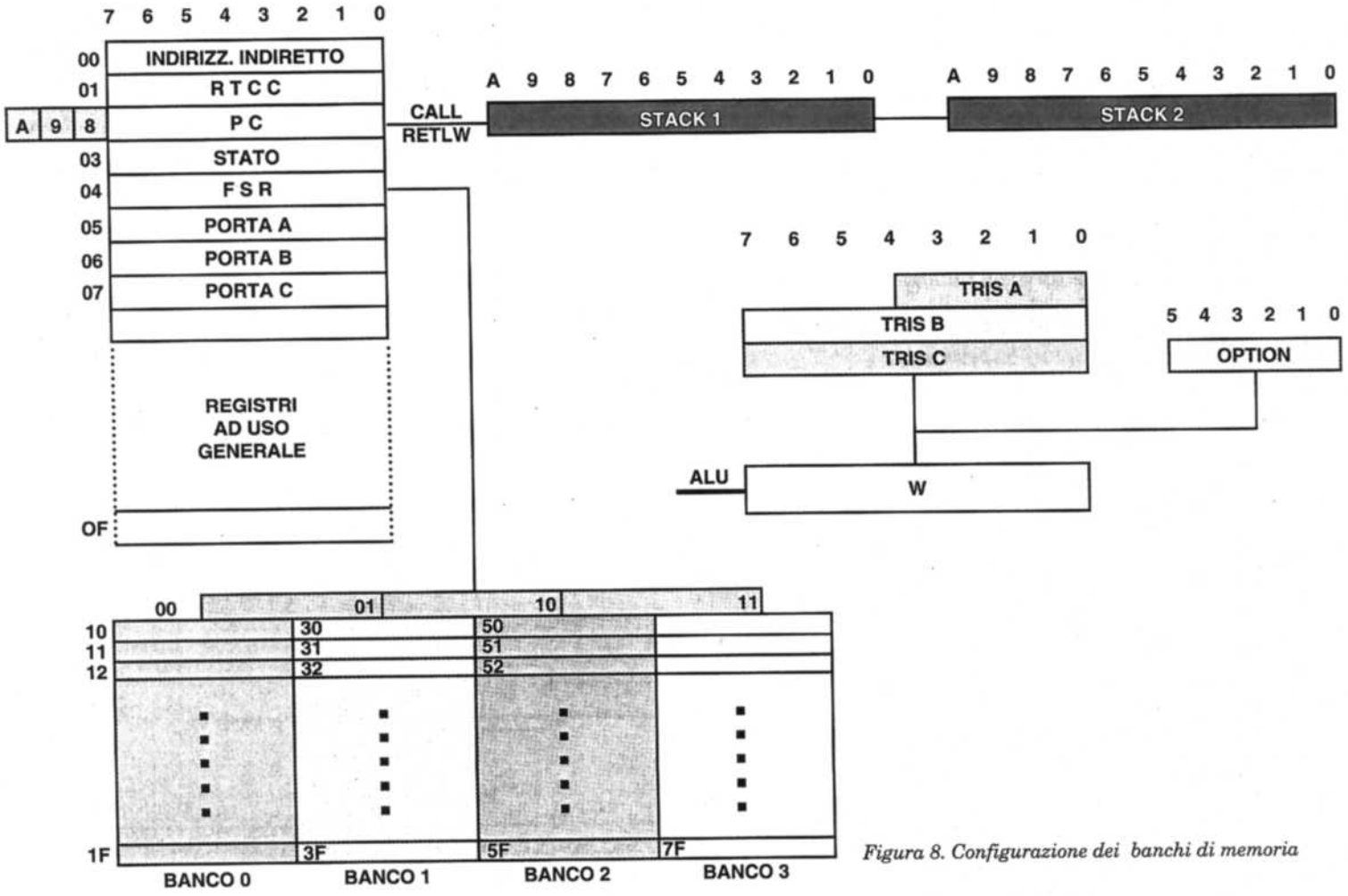



*Figura 8. Configurazione dei banchi di memoria*

È semplice intuire che quando il segnale sul clock del flip-flop in basso ha un fronte ascendente, il flip-flop memorizza sull'uscita Q il valore letto sull'ingresso D. Supponiamo che questo sia 1 e, quindi, Q si porta a 1 e il P-MOS viene interdetto e isola il pin di input dal positivo. Inoltre, essendo Q negato uguale a zero, anche il transistor N-MOS viene interdetto, isolando il pin di ingresso anche dal negativo.

A questo punto il pin in questione è stato programmato per funzionare da INPUT e qualsiasi valore tenti di scriverci via software viene ignorato.

Il dato di input potrà essere letto con un'istruzione che abiliti il three-state READ sul buffer di ingresso al BUS DATI. Supponiamo, invece, che il dato letto sia 0; allora i due transistor verranno abilitati a funzionare tramite le due porte OR e AND nel seguente modo: la porta AND darà un 1 in uscita quando il dato letto dal primo flip-flop è zero, mentre darà 1 nel caso contrario.

Quindi, il transistor N-MOS porterà il pin di input/output a massa se il dato immagazzinato nel primo flip-flop è 0 (pin configurato come uscita bassa). Viceversa, si attiverà il transistor P-MOS, portando il pin di input/output al positivo (pin configurato come uscita alta).

In definitiva, abbiamo visto, tornando alla Figura 2, che il registro TRIS_A (TRIS_B e TRIS_C) serve a configurare i vari pin come input oppure come output, anche separatamente (corrispondenza con il flip-flop in basso), mentre il registro F5 (F6 e F7) viene utilizzato per settare a 1 o a 0 i pin già configurati come uscita (corrispondenza con il flip-flop in alto) e leggere lo stato di quelli settati come input.

Il registro FSR serve per indirizzare indirettamente un altro registro e lo vedremo prossimamente con le istruzioni idonee. Il registro OPTION viene impiegato per attivare o disattivare il prescaler, per variarne la scala di riduzione e per decidere se il segnale di RTCC deve essere valutato sul fronte di discesa oppure su quello di salita.

In Figura 7 possiamo vedere questo registro bit per bit. Il registro PRESCALER è un registro dedicato al conteggio di impulsi che possono arrivarvi sia dal clock interno che da un pin apposito, il RTCC (Real Time Clock Counter).

Scopo del registro è quello di dividere gli impulsi in multipli di 2, fatto molto interessante se si pensa, ad esempio, di sfruttare il PIC come frequenzimetro o come lettore di rapporti impulso/pausa.

Il registro WATCHDOG, invece, se attivato, poiché gestito via software, consente di determinare se la CPU si è "bloccata" e, in tale situazione, ha il compito di resettare il tutto.

Questa funzione è molto utile specie in impianti dove il funzionamento di un certo circuito deve essere garantito in ogni momento. Infine, è presente un registro detto di CONFIGURAZIONE che non può essere scritto, ma solamente letto e che dà informazioni quali il device, il tipo di oscillatore e altre.

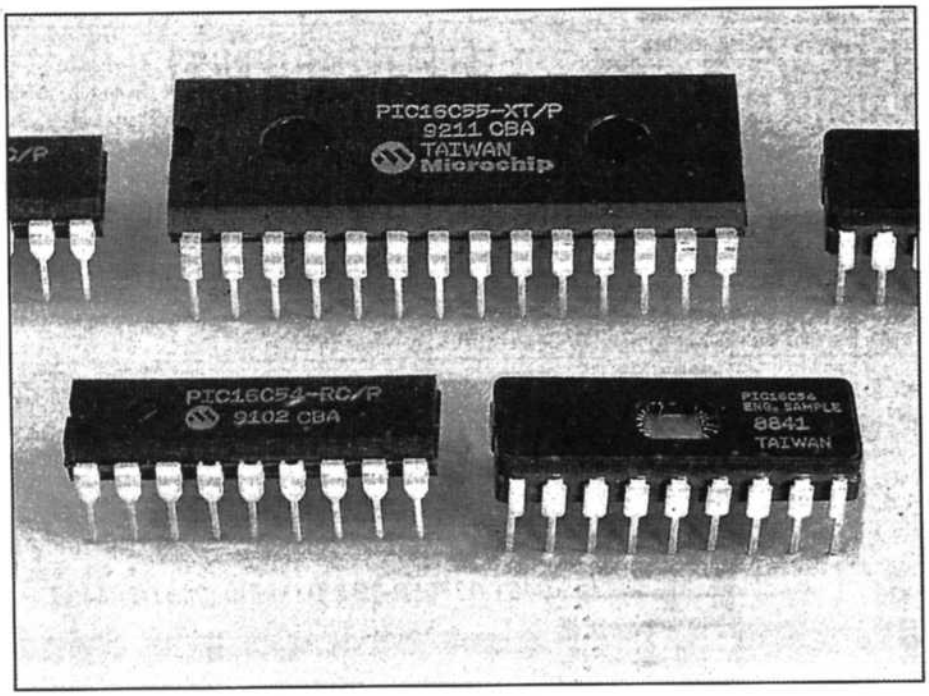
Per concludere la spiegazione sui PIC, ne riportiamo in Figura 8 la struttura interna in relazione ai vari banchi di memoria possibili.

*- continua -*

# CORSO DI PROGRAMMAZIONE PER IL CONTROLLER PIC16C5X

***Nella prima parte della descrizione di questi controller abbiamo iniziato l'analisi della struttura interna e dei registri principali di questo interessante famiglia di controller; questo mese vedremo le istruzioni del PIC, come vengono codificate e quali output producono.***

*di Andrea Sbrana IW5CBO - 2ª Parte*




Le prime nozioni da tenere presente quando ci si accinge a scoprire un nuovo controller sono la sua struttura e le sue istruzioni, per poter estrarre da esso il migliore dei risultati.

Per prima cosa notiamo che anche se il PIC è un controllore ad 8 bit, le istruzioni vengono codificate, insieme agli operandi, in un'area di dodici bit.

## Le istruzioni

Ogni istruzione ha una codifica, un codice mnemonico, l'eventuale operando e può o meno modificare lo stato del registro di STATO visto nella puntata precedente.

La prima operazione che prendiamo in esame è la NOP, ovvero la "no operation", cioè quella operazione che non modifica in alcun modo lo stato del PIC.

La domanda che molti già si porranno è questa: perché inserire una istruzione che non fa niente? Vedremo in seguito che per attese e per ogni tipo di timer tale istruzione deve essere obbligatoriamente usata.

La sua codifica vede tutti i bit a zero (000h), non ha operandi e non produce cambi di stato, ma semplicemente fa passare un ciclo in più durante l'esecuzione di un programma.

Anche l'istruzione MOVWF non altera il registro di stato del PIC, ma modifica il registro f (cioè il suo operando) immagazzinandovi ciò che trova nel registro W. Il numero di f potrà variare, a seconda del device, da zero a un massimo 1Fh.

Questa istruzione può essere utile ad esempio per settare un registro a un ben definito valore iniziale prima di un determinato conto.

Ci sono poi due istruzioni che resettano il contenuto di W e di un registro f dato come operando e sono rispettivamente CLRW (Clear W) e CLRF (Clear f). La chiamata a entrambe fa in modo che venga modificato il bit numero 2 del registro di STATO.

Ricordiamo che questo bit viene settato a 1 se il risultato di una istruzione è zero.

La codifica per CLRW è 040h, mentre per CLRF è 06fh.

Per valutare le successive istruzioni, fissiamo ora alcune notazioni: f è sempre considerato il numero di un registro e potrà variare tra 0 e 1fh, d invece indica la destinazione del risultato dell'operazione, che potrà essere W se d=0, oppure f se d=1.

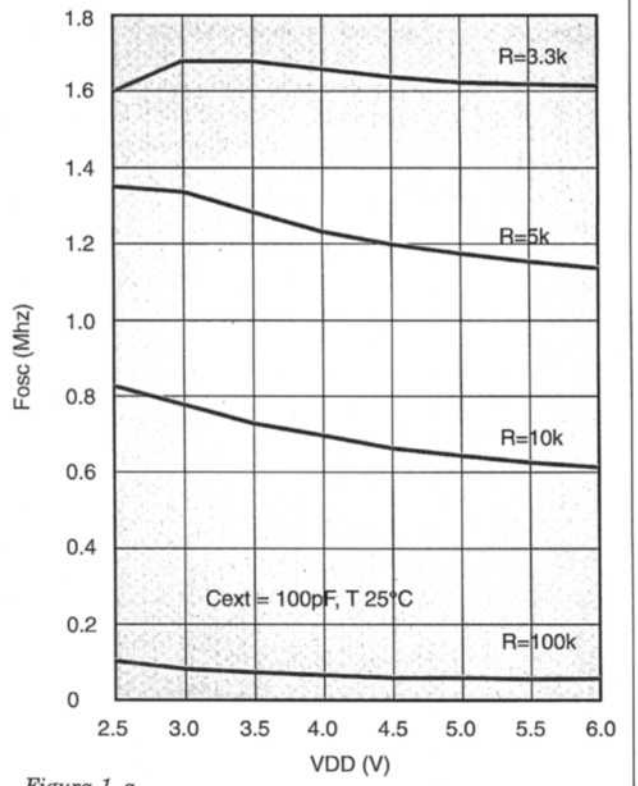



*Figura 1. a*

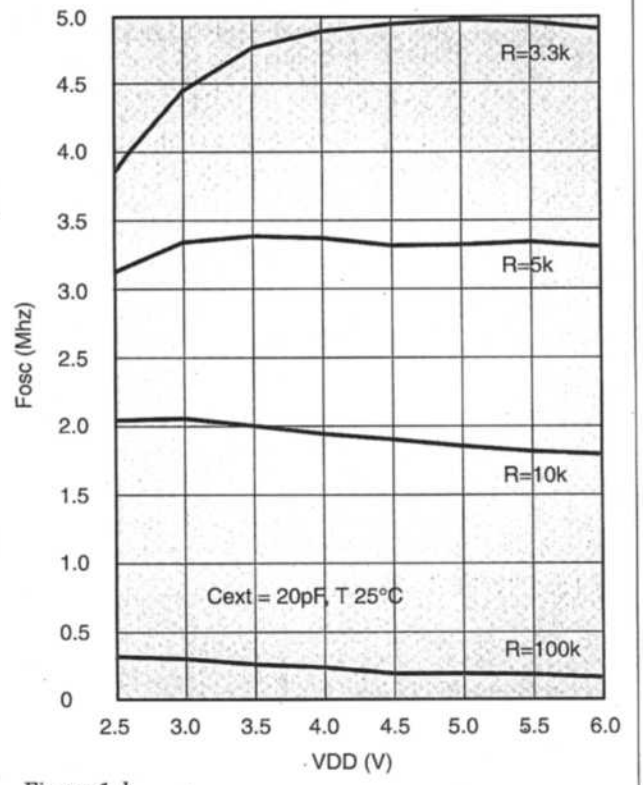
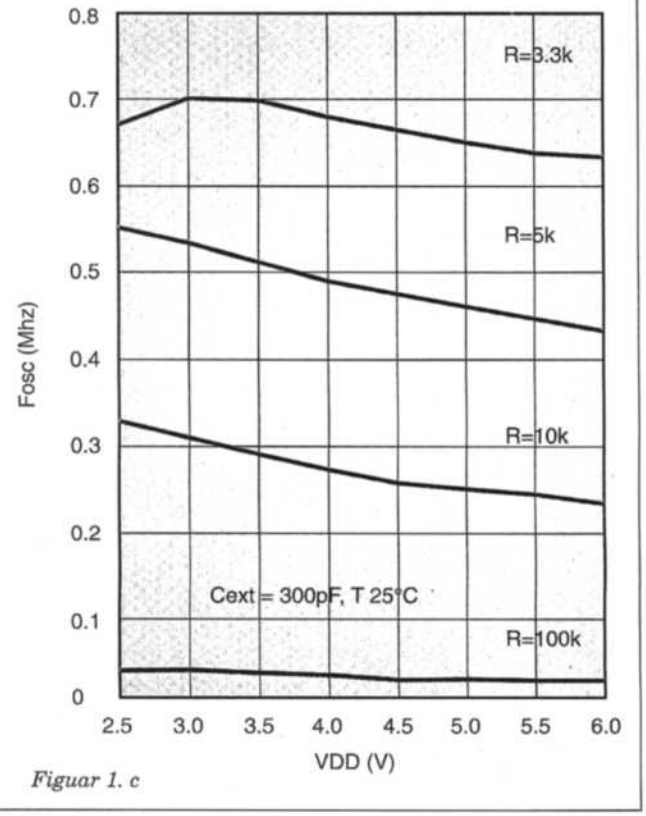



*Figura 1. b*




*Figuar 1. c*

| Cext | Rext | Average Fosc @ 5V, 25° C | |
|---|---|---|---|
| 20 pl | 3.3k | 4.71 MHz | ± 28% |
| | 5k | 3.31 MHz | ± 25% |
| | 10k | 1.91 MHz | ± 24% |
| | 100k | 207.76 KMz | ± 39% |
| 100pl | 3.3k | 1.65 MHz | ± 18% |
| | 5k | 1.23 MHz | ± 21% |
| | 10k | 711.54KHz | ± 18% |
| | 100k | 75.62 KHz | ± 28% |
| 300pl | 3.3k | 672.78 KHz | ±14% |
| | 5k | 489.49 KHz | ± 13% |
| | 10k | 275.73 KHz | ± 13% |
| | 100k | 28.12 KHz | ± 23% |

*Figura 2*

*Figura 1: a),b),c) slittamento in frequenza dei circuiti oscillanti RC*
*Figura 2: Valori di R e C per ottenere frequenze standard*

La prima istruzione a due operandi è la SUBWF (SUBtract W From).

Il primo operando indica il registro da cui sottrarre W e il secondo la destinazione: è possibile, infatti, sottrarre W ad esempio dal registro 09h e decidere se riscrivere il risultato in 09h oppure in W.

La codifica di SUBWF è 08Fh e a istruzione avvenuta vengono modificati i bit C, DC e Z del registro di STATO.

La DECF (DECrement f), invece, decrementa il valore che trova nel registro f e pone il risultato in f o in W.

Se il valore iniziale era zero, dopo questa operazione nel registro troveremo FFh.

Il codice della DECF è 0CFh e anche questa modifica il registro di STATO nel bit 2: può essere usata per introdurre dei cicli andando poi a testare il bit 2 di questo registro per vedere se si è arrivati a zero.

## Le istruzioni "booleane"

Esistono due tipiche istruzioni da matematica booleana e cioè la IORWF (Inclusive OR W and f) e la ANDWF (AND W and f) che, rispettivamente eseguono l'operazione di or e di and fra W e i registri indicati come operandi.

Il bit 2 del registro di STATO viene modificato in conseguenza.

Le codifiche per le due istruzioni sono, rispettivamente, 10Fh e 14Fh.

Come le precedenti, anche la XORWF (eXclusive OR W and f) compie un'operazione booleana e cioè l'or esclusivo tra W e il registro f.

Il suo codice operativo è 18Fh e anch'essa modifica il registro di STATO.

La ADDWF (ADD W and f) somma il valore di W a quello contenuto nel registro f e lo deposita in d.

Poiché possono intervenire problemi di overflow, vengono interessati i bit C, DC e Z del registro di STATO.

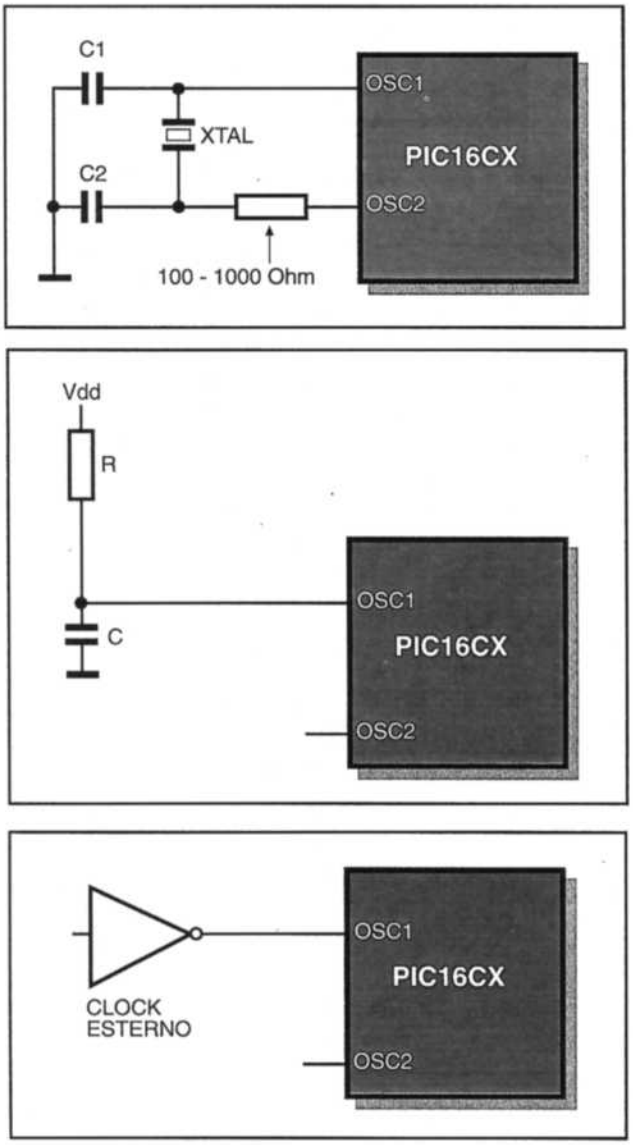
Il codice della ADDWF è 1CFh.

Analizziamo ora un'altra istruzione che "sposta" i contenuti dei registri, e cioè la MOVF (MOVe f). Con essa, il valore contenuto nel registro f viene immagazzinato nella destinazione d.

Poichè la destinazione può essere esclusivamente o il registro stesso o W, nel primo caso si ha l'effetto di una NOP (f viene riscritto con il suo valore).

Il codice di questa istruzione è 20Fh.

Con la COMF (COMplement f) viene




### OSCILLATORI AL CRISTALLO

| Osc Type | Freq | C1 | C2 |
|---|---|---|---|
| LP | 32 KHz | 15pF | 15pF |
| XT | 100 KHZ | 15 - 30 pF | 200 - 300 pF |
| | 200 KHz | 15 - 20 pF | 100 - 200 pF |
| | 455 KHz | 15 - 20 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 - 30 pF | 15 - 30 pF |
| | 8 MHZ | 15 pF | 15 pF |
| | 20 MHz | 15 pF | 15 pF |

### RISONATORI CERAMICI

| Oscillatori Type | Risonatori Frequenza | Capacità C1 = C2 |
|---|---|---|
| XT | 455 KHz | 150 - 330 pF |
| | 2.0 MHz | 20 - 330 pF |
| | 4.0 MHz | 20 - 330 pF |
| HS | 8.0 MHz | 20 - 200 pF |

*Figura 3. Valori dei componenti in realazione alla frequenza e al tipo di oscillatore*

eseguito il complemento del registro f, cioè vengono sostituiti gli 1 con gli 0 e viceversa.

Anche per questa istruzione viene influenzato il bit2 del registro di STATO. Il codice operativo della COMF è 24Fh.

L'istruzione complementare alla DECF vista precedentemente è la INCF (INCrement f): rispetta tutte le caratteristiche viste per la DECF, ma invece di decrementare il valore di f, lo incrementa di uno.

Il suo codice operativo è 28Fh.

Una istruzione molto importante e usata sempre per la creazione di cicli è la DECFSZ (DECrement f Sip se Zero). In pratica esegue un decremento unitario come la DEC, ma al termine controlla se il risultato è zero.

Se ciò avviene, l'istruzione seguente nel programma viene saltata (skipped) e si passa a quella successiva.

È intuibile che in un ciclo si debba sempre testare se il conteggio è giunto alla fine o meno e questa istruzione esegue proprio questa operazione.

Il suo codice è 2CFh.

**Le istruzioni orientate al byte**

| | | | | |
|---|---|---|---|---|
| 0000 0000 0000 | NOP | - | - | None |
| 0000 001F FFFF | MOVWF | f | f=W | None |
| 0000 0100 0000 | CLRW | - | W=0 | Z |
| 0000 011F FFFF | CLRF | f | f=0 | Z |
| 0000 10DF FFFF | SUBWF | f,d | d=f-W | C,DC,Z |
| 0000 11DF FFFF | DECF | f,d | d=f-1 | Z |
| 0001 00DF FFFF | IORWF | f,d | d=W or f | Z |
| 0001 01DF FFFF | ANDWF | f,d | d=W & f | Z |
| 0001 10DF FFFF | XORWF | f,d | d=W or-ex f | Z |
| 0001 11DF FFFF | ADDWF | f,d | d=W+f | C,DC,Z |
| 0010 00DF FFFF | MOVF | f,d | d=f | Z |
| 0010 01DF FFFF | COMF | f,d | d=f comp. | Z |
| 0010 10DF FFFF | INCF | f,d | d=f+1 | Z |
| 0010 11DF FFFF | DECFSZ | f,d | d=f-1, salta se zero | None |
| 0011 00DF FFFF | RRF | f,d | d(n-1)=f(n),d(7)=C,C=f(0) | C |
| 0011 01DF FFFF | RLF | f,d | d(n+1)=f(n),d(0)=C,C=f(7) | C |
| 0011 10DF FFFF | SWAPF | f,d | d=f(4-7)-f(03) | None |
| 0011 11DF FFFF | INCFSZ | f,d | d=f+1, salta se zero | None |

*Tabella 1*

***"I Pic, in virtù del fatto che sono processori "risc" dispongono di poche istruzioni, comunque sufficienti a svolgere operazioni anche complesse"***

### La manipolazione dei byte

Ci sono poi tre istruzioni che manipolano il byte del registro f e cioè la RRF (Rotate Right f), la RLF (Rotate Left f) e la SWAPF (SWAP halves f).

La prima ruota i bit del registro f verso destra, cioè in pratica li "shifta" di una posizione: il bit ottavo va al posto del settimo, il settimo al posto del sesto e così via fino al primo che va nel bit C del registro di STATO.

La RLF shifta anch'essa, ma verso sinistra con le stesse modalità operative della RRF.

La SWAPF, invece, scambia di posto i due nibble del byte da trattare.

I codici operativi delle tre istruzioni sono rispettivamente 30Fh, 34Fh e 38Fh.

Infine, la INCFSZ (INCrement f Skip se Zero) ha le stesse caratteristiche viste per la DECFSZ, ma questa volta l'operazione sul valore del registro è di incremento e non di decremento, utile per cicli con conteggio positivo.

Il suo codice operativo è 3CFh.

Abbiamo visto così tutte le istruzioni orientate al byte che sono presenti in Tabella 1.

In Tabella 2, invece, troviamo le sole quattro istruzioni orientate al bit.

Precisiamo che f indica il solito registro, mentre b il numero del bit su cui operare. b potrà valere da 0 a 7 essendo il registro f un 8 bit.

La prima istruzione che valutiamo è la BCF (Bit Clear f) che pone a zero il bit b del registro f.

Questa istruzione è molto usata per disattivare un'uscita se, ad esempio, il registro è il 5, il 6 o il 7.

Il suo codice operativo è 4BFh.

La complementare della BCF è la BSF (Bit Set f), che pone a 1 il bit b del registro f. Anche questa è molto utile per attivare le uscite del PIC.

Il suo codice operativo è 5BFh.

Ci sono ora due istruzioni che testano il valore di un singolo bit e saltano l'istruzione successiva se tale bit è a zero (BTFSC cioè Bit Test f Skip se Clear) oppure a uno (BTFSS cioè Bit Test f Skip se Set).

I loro codici operativi sono rispettivamente 6BFh e 7BFh.

Al contrario delle due precedenti, queste istruzioni di test consentono di verificare il livello logico che ho su una porta di input.

Le ultime operazioni che prendiamo in esame sono quelle visibili in Tabella 3 e cioè quelle relative all'input di dati da programma (K indica il valore espresso in un byte) e al controllo del PIC.

**Le istruzioni orientale al bit**

| | | | |
|---|---|---|---|
| 0100 BBBF FFFF BCF | f,b | f(b)=0 | None |
| 0101 BBBF FFFF BSF | f,b | f(b)=1 | None |
| 0110 BBBF FFFF BTFSC | f,b | Salta se bit(b) di f è 0 | None |
| 0111 BBBF FFFF BTFSS | f,b | Salta se bit(b) di f è 1 | None |

*Tabella 2*

**Le istruzioni relative all'input di dati e di controllo del PIC**

| | | | |
|---|---|---|---|
| 0000 0000 0010 OPTION | - | OPTION=W | None |
| 0000 0000 0011 SLEEP | - | WDT=0,stop oscillator | TO,PD |
| 0000 0000 0100 CLRWDT | - | WDT=0 and prescaler | TO,PD |
| 0000 0000 0FFF TRIS | f | f=W | None |
| 1000 KKKK KKKK RETLW | k | W=k, PC=STACK | None |
| 1001 KKKK KKKK CALL | k | STACK=PC+1, PC=k | None |
| 101K KKKK KKKK GOTO | k | PC=k | None |
| 1100 KKKK KKKK MOVLW | k | W=k | None |
| 1101 KKKK KKKK IORLW | k | W=k OR W | Z |
| 1110 KKKK KKKK ANDLW | k | W=k & W | Z |
| 1111 KKKK KKKK XORLW | k | W=k or-ex W | Z |

*Tabella 3*

L'operazione OPTION (load OPTION register) carica nel registro OPTION il contenuto di W.

Quindi questa istruzione sarà utile in fase di inizializzazione per settare correttamente le opzioni del PIC.

Il suo codice operativo è 002h.

L'istruzione SLEEP (go into standby mode) pone a zero il WDT (WatchDog Timer) e, quindi, ferma l'oscillatore, ottenendo un consumo totale del chip inferiore a 15 µA.

Il suo codice operativo è 003h.

Anche la CLRWDT (CLeaR WAtchDog Timer) pone uno zero nel WDT, ma questa volta l'oscillatore non viene bloccato e il programma prosegue.

È molto utile per introdurre un controllo sull'effettivo funzionamento del PIC: se il programma ripetutamente azzera il WDT, il PIC funziona normalmente.

Quando questo azzeramento viene a mancare, significa che per qualche motivo esterno il PIC non procede più con il programma regolare e allora interviene un reset interno automatico.

Il codice operativo della CLRWDT è 004h.

L'istruzione TRIS (TRIstate port) consente di programmare i singoli pin come ingressi o come uscite: scrivendo un 1 su di un bit corrispondente ad esempio al registro 6 bit 0 (porta B pin 0), si dice al PIC che quel pin deve essere di ingresso.

Al contrario, scrivendoci 0 diciamo che quel pin deve essere di uscita.

Nella scorsa puntata abbiamo fatto vedere come sono fisicamente le porte del PIC e con questa operazione provate a vedere se effettivamente le cose funzionano come avevamo preannunciato.

Il codice operativo è 00Fh.

Le due istruzioni seguenti consentono di introdurre delle subroutine e sono la RETLW (RETurn, place k in W) e CALL (CALL subroutine).

Entrambe modificano i valori del PC e degli stack come spiegato nella scorsa puntata. I loro codici operativi sono rispettivamente 8KKh e 9KKh.

L'istruzione GOTO (GO TO address), invece, offre la possibilità di saltare incodizionatamente a un indirizzo del programma scelto a piacere.

Il suo codice operativo è AKKh.

Con l'operazione MOVLW (MOVe Literal to W), il registro W viene "caricato" con il valore k.

Può essere molto utile ad esempio per inizializzare dei registri con dei valori inseriti nel programma come dati. Il suo codice operativo è CKKh.

Le ultime tre operazioni che vedremo sono di tipo booleano e cioè IORLW (Inclusive OR Literal and W), ANDLW (AND Literal and W), XORLW (eXclusive OR Literal and W).

Le operazioni booleane svolte da queste istruzioni sono le stesse viste per le IORWF, ANDWF e XORWF con la differenza che la destinazione è sempre W e l'altro operando è un valore impostabile durante l'operazione stessa.

I loro codici operativi sono, rispettivamente, DKKh, EKKh e FKKh.

Abbiamo così valutato, una per una, tutte le possibili istruzioni per il PIC16C5X.

## L'oscillatore

Aggiungiamo ora alcuni dati che sono necessari per l'impiego di questo chip e che nella scorsa puntata non abbiamo avuto spazio per mostrarvi.

La scelta di un tipo di oscillatore al posto di un altro deve essere ben motivata e vedremo i pregi e i difetti di ciascuno di questi.

Diciamo innanzitutto che i tipi di oscillatori sono 4: XT, HS, RC, LP.

La versione finestrata del PIC ha la possibilità di essere settata, tramite il software di programmazione, per funzionare con qualsiasi tipo di oscillatore, mentre la versione OTP (One Time Programmable) viene settata dal fabbricante stesso e, quindi, non più modificabile dall'utente.

La differenza fondamentale fra gli oscillatori controllati con quarzo oppure con risuonatore ceramico e quelli RC è senza dubbio la stabilità in frequenza, mentre la secondaria è il costo di produzione.

In Figura 1a, 1b, 1c potete vedere lo slittamento di frequenza in funzione della temperatura con un circuito oscillante di tipo RC. In Figura 2, invece, sono riportati alcuni valori di R e di C per ottenere alcune frequenze standard.

In Figura 3 possiamo vedere le connessioni con i vari tipi di oscillatore.

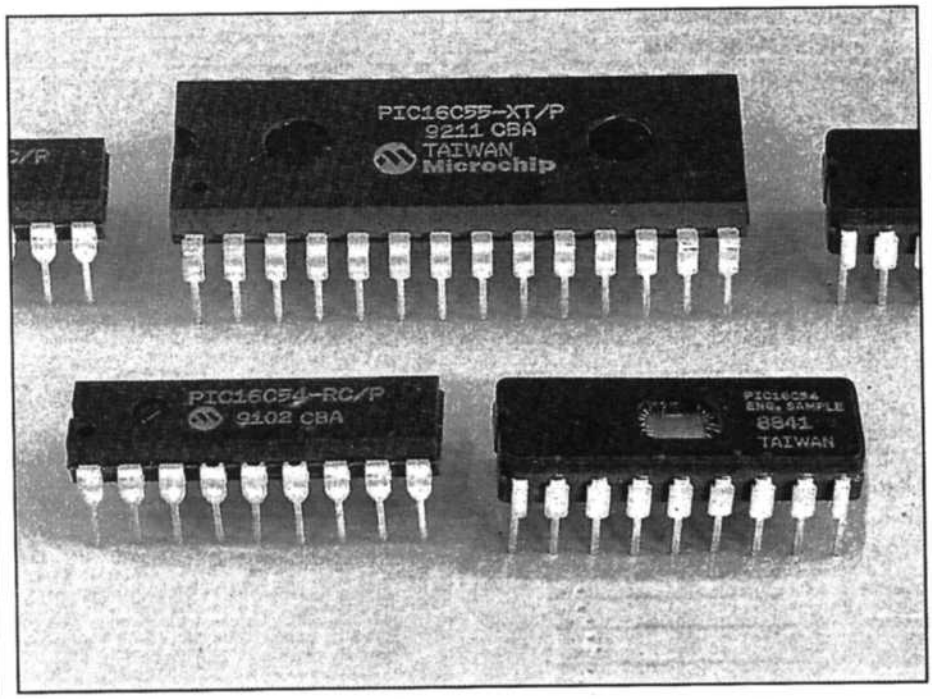
Il prossimo mese proveremo a scrivere qualche programma e a commentarlo per far capire la filosofia della programmazione di una CPU-like-RISC.

*- continua -*

# PIC16C5X: IMPARIAMO A PROGRAMMARLO

***Dopo aver visto la struttura interna del PIC e le sue istruzioni, iniziamo a presentare alcuni programmi e routines che sono di interesse universale e che, per la loro semplicità, riescono a far capire il funzionamento del chip, unendo una struttura hardware a una software.***

*Andrea Sbrana IW5CBO - 3ª parte*




Nella stesura di un programma, per semplificare le operazioni di codifica e di modifica successiva è necessario seguire uno schema logico abbastanza preciso.
Incominciando dall'inizio, la prima riga del programma deve contenere la direttiva TITLE, come quella dell'esempio seguente:

TITLE 'Programma di prova per PROGETTO'.

Dopo la scritta TITLE, potrete dare un titolo a piacere al vostro programma, sempre però inserito fra due apici.

La seconda riga indica al compilatore quale device state usando e quale formato di file in uscita volete:

list F=INHX16,P=16C54

Nel nostro caso abbiamo un PIC16C54 e un object file in formato 16 bit word. Inoltre, la prima dichiarazione list implica che deve essere generato un file .LST, che servirà successivamente per un miglior debug del programma.

A questo punto, possiamo iniziare a scrivere il programma, ma è conveniente incominciare a fissare delle costanti e rinominare dei registri per una comodità di scrittura e di interpretazione:

```
TIM0    EQU 1AH     ;numero
                    ripetizioni
                    per delay 10 mS
F5      EQU 5       ;porta A
F6      EQU 6       ;porta B
F7      EQU 7       ;utilità gener.
F8      EQU 8
F9      EQU 9
F10     EQU 0AH
F11     EQU 0BH
F12     EQU 0CH
F13     EQU 0DH
F14     EQU 0EH
F15     EQU 0FH
F16     EQU 10H
F17     EQU 11H
F18     EQU 12H
F19     EQU 13H
F20     EQU 14H
F21     EQU 15H
F22     EQU 16H
F23     EQU 17H
F24     EQU 18H
F25     EQU 19H
F26     EQU 1AH
F27     EQU 1BH
F28     EQU 1CH
F29     EQU 1DH     ;per ritardo 1 S
F30     EQU 1EH     ;per ritardo 10mS
F31     EQU 1FH     ;per ritardo 10mS
```

Abbiamo così definito una costante che si chiama TIM0 con assegnato il valore 1Ah; per ogni registro, inoltre, abbiamo dato una ridenominazione più facile da interpretare: sicuramente è più istintivo chiamare un registro F31 che 1Fh. Ogni frase scritta dopo il punto e virgola viene interpretata dal compilatore come commento e, quindi, non valutata.

## Il programma

Possiamo così passare al programma vero e proprio:

ORG 0

è un'altra direttiva del compilatore e definisce l'origine del programma stesso.

OPTION

Option, senza aver scritto niente in W, indica che non si vogliono utilizzare il WATCHDOG e il prescaler.

In genere, a questo punto del programma, vengono inserite le subroutine di uso più frequente, come quella per il ritardo di 10 mS riportata di seguito:

```
;ROUTINE DI RITARDO 10 mS

DELAY
    MOVLW TIM0      ;carico il
                    valore di
                    TIM0 (1Ah) in W
    MOVWF F30       ;carico W in
                    F30 (1Eh)
DEL0
    MOVWF F31       ;carico W in
                    F31 (1Fh)
DEL1
    NOP             ;nessuna
                    operazione
    NOP             ;nessuna
                    operazione
    NOP             ;nessuna
                    operazione
    DECFSZ F31      ;decremento
                    F31, salto se
                    risultato è 0
    GOTO DEL1       ;vado a DEL1
    DECFSZ F30      ;decremento
                    F30, salto se
                    risultato è 0
    GOTO DEL0       ;vado a DEL0
    RETLW 0         ;ritorna da
                    subroutine
```

La scritta DELAY è una label, come pure DEL0 e DEL1.

Queste ci permettono di saltare da un punto all'altro del programma con le semplici istruzioni GOTO e CALL e di tornare, alla fine della routine, al punto della chiamata con RETLW.

Cerchiamo di capire come funziona questa routine di ritardo; per semplicità in Figura 1 è visibile il diagramma a blocchi di questa funzione.

Inizialmente, i due registri F30 e F31 vengono caricati con il valore 1Ah, che corrisponde a 26 decimale.

Vengono poi eseguite tre operazioni di tipo NOP, cioè di quelle che non modificano lo stato del PIC, ma che servono solo a far passare il tempo di tre istruzioni (vedi puntate precedenti).

Successivamente, con una sola operazione viene decrementato il registro F31 e il PIC si chiede se il risultato è uguale a zero.

In caso contrario, la DECFSZ non "skippa" a due istruzioni seguenti, ma esegue la successiva e torna a DEL1.

In questo modo, dopo 26 cicli sicuramente F31 è uguale a zero.

Allora il registro F31 viene decrementato e sottoposto allo stesso test dell'uguaglianza con zero.

Similmente alla situazione precedente, se F30 non vale zero, si torna a DEL0 dove viene ricaricato F31 con 1Ah e si riparte come prima.

Dopo 26 cicli, questa volta calcolati solo su F30, si trova l'istruzione RETLW, che ci fa tornare al punto in cui avevamo invocato una CALL DELAY.

Vogliamo ora capire quanto tempo è stato impiegato dal PIC per tutto questo lavoro, e lo calcoliamo in questo modo: 1 operazione consiste nel caricare inizialmente F30.

L'operazione di caricare il registro F31, invece, viene fatta per ben 26 volte.

Le tre NOP sono invocate per ben 26*26*3 volte, e la DECFSZ F31 insieme alla GOTO DEL1 per altrettante 26*26 volte.

La DECFSZ F30, invece, per 26 volte, come pure la GOTO DEL0.

Sommando tutti questi numeri, otteniamo 3.435 operazioni.

Dobbiamo ora ricordarci che il PIC suddivide al suo interno il clock per quattro, quindi ogni 4 colpi di clock esegue un'operazione.

Allora i colpi di clock per 3.435 operazioni sono 4*3.435 = 13.740.

Se il clock allora fosse stato di 1 MHz il tempo fra un colpo di clock e il successivo sarebbe stato di 1 microsecondo (1/1.000.000) e, quindi, moltiplicato per 13.740 avrebbe dato circa 14 millisecondi di ritardo.

Ovviamente a noi non interessava avere 10 millisecondi esatti, ma nel caso in cui a qualcuno serva un tempo di riferimento molto preciso, consigliamo di usare un quarzo a circa 4 MHz, che fra l'altro facilita poi la divisione dei colpi di clock per 4.

Utilizzando poi questa routine come riferimento, è possibile ottenere altri ritardi.

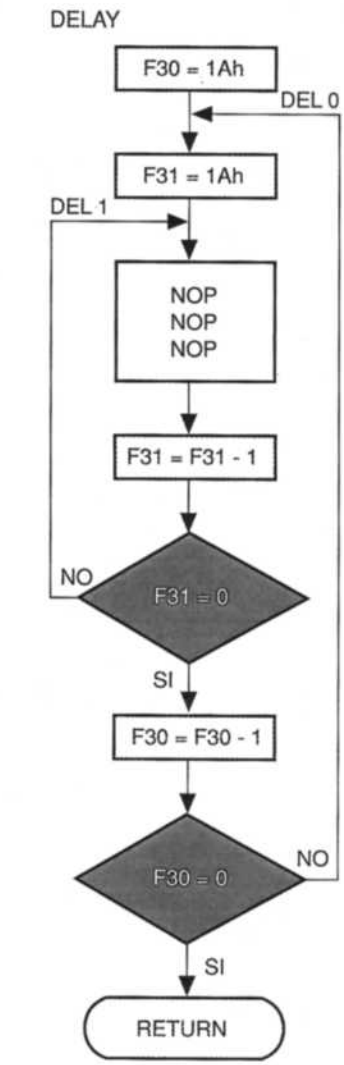



*Figura 1. Diagramma a blocchi della routine di ritardo 10 ms*

In un nostro progetto, era utile avere un ritardo di circa un secondo per l'azionamento di un relè e per far lampeggiare un Led. Vediamo come è possibile farlo con poche semplici istruzioni:

```
;ROUTINE DI RITARDO 1S
DELAY1
   MOVLW 48H   ;carico il valore di (48h) in W
   MOVWF F29   ;carico W in F29 (1Eh)
DEL2S
   CALL DELAY
   DECFSZ F29
   GOTO DEL2S
   RETLW 0
```

In questo modo viene chiamata la routine DELAY per 72 volte, introducendo così un ritardo di circa 13,5*72=0,972 Ms cioè di un secondo. La nuova istruzione che è stata introdotta è la CALL.

Da notare che sia la routine DELAY che la DELAY1, vengono invocate con una CALL, quindi durante la chiamata a quella più esterna, cioè la DELAY1, non è possibile fare altre CALL: ricordiamo, infatti, che il numero massimo di chiamate a subroutine possibile è di due.

Accingiamoci ora a codificare il programma vero e proprio:

```
MAIN
    MOVLW 00FH
    TRIS 6          ;PORTA -B- B0..B3
                    IN B4..B7 OUT
    MOVLW 0FFH
    MOVWF F6
    CLRW            ;azzera W
    TRIS 5          ;PORTA -A- OUT
    MOVWF F5        ;tutti i pin di A a zero
MAIN1
    BTFSC F5,0      ;testa porta A bit 0,
                    skippa se zero
    GOTO SETTA      ;vai a SETTA
    BSF F5,0        ;setta a 1 bit 0 di porta A
    CALL DELAY1     ;ritardo di un secondo
    GOTO MAIN1      ;torna a MAIN1
SETTA
    BCF F5,0        ;setta a 0 bit 0 di porta A
    CALL DELAY1
    GOTO MAIN1      ;torna a MAIN1
```

Per standardizzare tutti i programmi, è bene che inizino sempre con la label MAIN.

Le prime operazioni da effettuare sono il settaggio delle porte, cioè la direzione scelta (input oppure output) e i loro valori iniziali.

Il programma molto semplice che abbiamo ora realizzato accende e spegne un'uscita della porta A con frequenza di un hertz circa.

In Figura 2 potete vedere il diagramma a blocchi.

Notate che se anche la porta A è stata configurata come uscita, è possibile leggere il suo stato e vedere se è a livello basso oppure alto.

Alla fine del programma è possibile inserire dei dati personali per l'identificazione: tenete conto, però, che questi dati vengono inseriti nella EPROM del PIC, quindi portano via dello spazio potenzialmente utilizzabile per lo sviluppo del programma.

```
DATA " LAMPEGGIO DI UN LED "
DATA " L.G.S. "
DATA " BY Andrea Sbrana "
```

La fine di un programma viene definita dalle seguenti tre righe:

```
ORG 01FFH
GOTO MAIN
END
```

La prima riga indica al PIC la cella di memoria da cui partire a leggere le istruzioni: nel caso del PIC16C54 e del PIC16C55 è obbligatorio dare 01FFh come origine (sarebbe l'ultimo indirizzo utile della EPROM).

Poiché ovviamente non è possibile continuare incrementando tale indirizzo, l'istruzione da dare successivamente sarà per forza di cose una GOTO MAIN, se avete deciso di iniziare con la label MAIN.

Infine, l'ultima riga del programma deve avere la direttiva END, per comunicare all'assemblatore che il programma da assemblare è terminato.

## Il programma per la chiave elettronica

Ora che ci siamo impratichiti con le istruzioni più comuni, vediamo un programma più complesso che utilizza dei piccoli trucchi per risolvere situazioni a prima vista molto difficoltose: il programma che è memorizzato nel PIC della chiave elettronica a microprocessore pubblicata su Progetto di alcuni mesi fa è visibile in Figura 3.

In Figura 4 troviamo lo schema a blocchi che ci aiuterà a capire meglio il funzionamento del dispositivo.

Come prima cosa da notare, vediamo che la routine che introduce il ritardo di circa 1/2 secondo non è stata fatta come la precedente e questo per un ben preciso motivo: supponiamo, infatti, di arrivare a un segmento di programma con una CALL e da qui voler attendere 1/2 secondo.

Allora con la vecchia soluzione avremmo avuto altre due CALL, una alla DELAY e l'altra alla DELAY, ma come abbiamo già detto questo nel PIC è impossibile perché i livelli di stack sono solamente due.

Dopo il MAIN notiamo un settaggio iniziale delle porte non molto diverso dal precedente, sebbene qui sia citata anche la porta C.

Da MAIN1 in poi il programma è attivo: la prima routine che chiama è quella per vedere se la chiave è stata inserita o no (KEYON).

Per far ciò vengono utilizzate delle "maschere", cioè delle istruzioni che ci consentono di accedere ai soli bit che ci interessano di un determinato byte.

Per essere certi di una risposta attendibile, vengono introdotti dei piccoli ritardi dopo ogni lettura.

Si noti che la routine termina depositando in W o zero, se la chiave risulta inserita, o uno altrimenti.

Al ritorno di questa routine, per verificare se la chiave è o meno inserita, si utilizza il seguente metodo: si carica

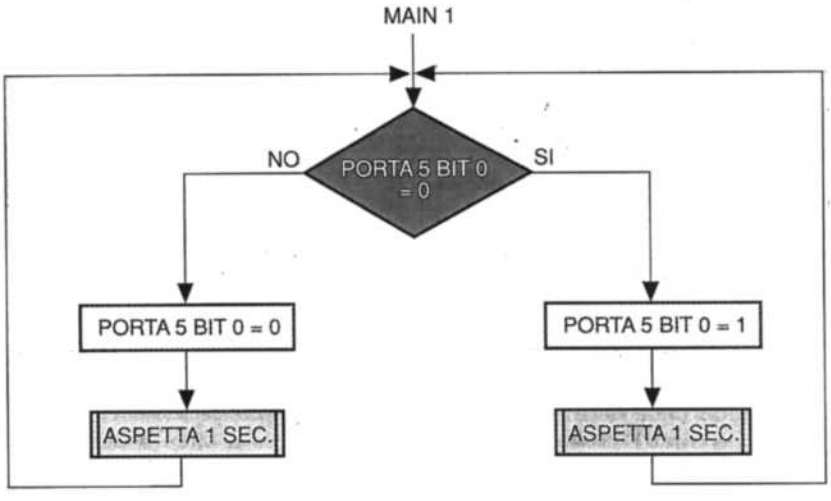



*Figura 2. Diagramma a blocchi per il lampeggio di un led*

```
TITLE 'KEY14IN
list F=INHX16,P=16C55

TIM0   EQU 79H  ;50 mS = 79H
TIM1   EQU 10H  ;1/2 SECONDO = 10H
RIP    EQU 1EH  ;NUMERO RIPETIZIONI ERRORE LAMPEGGIO=1EH
F5     EQU 5
F6     EQU 6
F7     EQU 7
F8     EQU 8
F9     EQU 9
F10    EQU 0AH  ;CODICE PORTA B
F11    EQU 0BH  ;CODICE PORTA C
F12    EQU 0CH  ;CONTATORE LAMPEGGIO ERRORE
F13    EQU 0DH  ;INIZIALIZZATO FFH PER RICONOSCIMENTO
                CHIAVE INSERITA
F14    EQU 0EH  ;INIZIALIZZATO FCH PER RICONOSCIMENTO
                CHIAVE INSERITA
F15    EQU 0FH  ;BUFFER PER RITORNO SUBROUTINE KEYON
F16    EQU 10H
F17    EQU 11H
F18    EQU 12H
F19    EQU 13H
F20    EQU 14H
F21    EQU 15H
F22    EQU 16H
F23    EQU 17H
F24    EQU 18H
F25    EQU 19H  ;RITARDO 1/2 SECONDO
F26    EQU 1AH  ;RITARDO 1/2 SECONDO
F27    EQU 1BH  ;RITARDO 1/2 SECONDO
F28    EQU 1CH
F29    EQU 1DH
F30    EQU 1EH  ;RITARDO 10mS
F31    EQU 1FH  ;RITARDO 10mS

       ORG 0

       OPTION

;ROUTINE DI RITARDO 10 mS
DELAY
       MOVLW TIM0
       MOVWF F30
DEL0
       MOVWF F31
DEL1
       NOP
       NOP
       NOP
       DECFSZ F31
       GOTO DEL1
       DECFSZ F30
       GOTO DEL0
       RETLW 0

;ROUTINE DI RITARDO 1/2 S
DELAY1
       MOVLW TIM1
       MOVWF F25
DEL2S
       MOVLW TIM0
       MOVWF F26
DEL2A
       MOVWF F27
DEL2B
       NOP
       NOP
       NOP
       DECFSZ F27
       GOTO DEL2B
       DECFSZ F26
       GOTO DEL2A
       DECFSZ F25
       GOTO DEL2S
       RETLW 0

KEYON                ;CONTROLLO SE LA CHIAVE E' INSERITA
       MOVF F7,0     ;LETTURA PORTA C
       SUBWF F13,0   ;CONFRONTO PORTA C
       BTFSS F3,2
       GOTO ASSEST
PORT2
       MOVLW 0FCH    ;MASCHERA RB7..RB2
       ANDWF F6,0
       SUBWF F14,0   ;CONFRONTO PORTA B
       BTFSC F3,2
       RETLW 1       ;CHIAVE NON INSERITA
       CALL DELAY
       MOVLW 0FCH    ;MASCHERA RB7..RB2
       ANDWF F6,0
       SUBWF F14,0   ;CONFRONTO PORTA B
       BTFSS F3,2
       RETLW 0       ;CHIAVE INSERITA
       RETLW 1       ;CHIAVE NON INSERITA
ASSEST
       CALL DELAY
       MOVF F7,0     ;LETTURA PORTA C
       SUBWF F13,0   ;CONFRONTO PORTA C
       BTFSS F3,2
       RETLW 0       ;CHIAVE INSERITA
       GOTO PORT2


PROG
       BSF F5,1      ;LED OFF SPENTO
       BCF F5,2      ;LED PROGRAM ACCESO
KEYIN
       CALL KEYON    ;ATTESA INSERIMENTO CHIAVE
       MOVWF F15
       BTFSC F15,0
       GOTO KEYIN    ;CHIAVE NON INSERITA
       CALL DELAY1   ;RITARDO ASSESTAMENTO CHIAVE
       CALL DELAY1
       MOVF F7,0
       MOVWF F11     ;MEMORIZZAZIONE INPUT PORTA C
       MOVF F6,0
       ANDWF F14,0
       MOVWF F10     ;MEMORIZZAZIONE INPUT PORTA B
       BSF F5,2      ;LED PROGRAM SPENTO
KEY1
       CALL KEYON    ;ATTESA DISINSERIMENTO CHIAVE
       MOVWF F15
       BTFSS F15,0
       GOTO KEY1     ;CHIAVE INSERITA
       BCF F5,1      ;LED OFF ACCESO
       GOTO MAIN1

CONFR                ;INIZIO CONFRONTO
       CALL DELAY1   ;RITARDO ASSESTAMENTO CHIAVE
```

```
    CALL DELAY1
    MOVF F7,0          ;LETTURA PORTA C
    SUBWF F11,0        ;CONFRONTO PORTA C
    BTFSS F3,2
    GOTO ERROR         ;CODICE ERRATO

    MOVLW 0FCH         ;MASCHERA RB7..RB2
    ANDWF F6,0
    SUBWF F10,0        ;CONFRONTO PORTA B
    BTFSS F3,2
    GOTO ERROR         ;CODICE ERRATO
    GOTO RELE

ERROR
    MOVLW RIP          ;NUMERO RIPETIZIONI LAMPEGGIO LED
    MOVWF F12
CICLO
    CALL DELAY1
    BCF F5,0           ;LED ON ACCESO
    BCF F5,1           ;LED OFF ACCESO
    CALL DELAY1
    BSF F5,0           ;LED ON SPENTO
    BSF F5,1           ;LED OFF SPENTO
    DECFSZ F12,1
    GOTO CICLO
    BTFSC F5,3
    GOTO CHON
    BSF F5,0           ;LED ON SPENTO
    BCF F5,1           ;LED OFF ACCESO
    GOTO MAIN1
CHON
    BCF F5,0           ;LED ON ACCESO
    BSF F5,1           ;LED OFF SPENTO
    GOTO MAIN1

RELE
    BTFSC F6,0         ;CONTROLLO SE J1 SETTATO A MASSA (S/R)
    GOTO IMPULS
    BTFSS F5,3         ;CONTROLLO STATO RELE
    GOTO SETTA
    BCF F5,3           ;RELE OFF
    BCF F5,1           ;LED OFF ACCESO
    BSF F5,0           ;LED ON SPENTO
SET
    CALL KEYON         ;ATTESA DISINSERIMENTO CHIAVE
    MOVWF F15
    BTFSS F15,0
    GOTO SET           ;CHIAVE INSERITA
    GOTO MAIN1         ;CHIAVE DISINSERITA
SETTA
    BSF F5,3           ;RELE ON
    BSF F5,1           ;LED OFF SPENTO
    BCF F5,0           ;LED ON ACCESO
    CALL KEYON         ;ATTESA DISINSERIMENTO CHIAVE
    MOVWF F15
    BTFSS F15,0
    GOTO SETTA         ;CHIAVE INSERITA
    GOTO MAIN1         ;CHIAVE DISINSERITA
IMPULS
    BSF F5,3           ;RELE ON
    BSF F5,1           ;LED OFF SPENTO
    BCF F5,0           ;LED ON ACCESO
    CALL DELAY1
    CALL DELAY1
    BCF F5,3           ;RELE OFF
```

```
CICLO1             ;LAMPEGGIO LED ON FINO A CHIAVE TOLTA
    CALL DELAY1
    BTFSC F5,0     ;CONTROLLO STATO LED ON
    GOTO CAMBIA
    BSF F5,0       ;LED ON SPENTO
    GOTO LAMP
CAMBIA
    BCF F5,0       ;LED ON ACCESO
LAMP
    CALL KEYON
    MOVWF F15
    BTFSS F15,0
    GOTO CICLO1    ;CHIAVE INSERITA
    BSF F5,0       ;LED ON SPENTO
    BCF F5,1       ;LED OFF ACCESO
    GOTO MAIN1

MAIN
    CLRW
    MOVLW 0FFH
    TRIS 7         ;PORTA -C- IN
    MOVLW 0FFH
    TRIS 6         ;PORTA -B- IN
    MOVLW 000H
    TRIS 5         ;PORTA -A- OUT
    CLRF F10       ;CODICE INIZIALE TUTTI I PIN  A MASSA
    CLRF F11
    MOVLW 05H
    MOVWF F5       ;SETTAGGIO INIZIALE PORTA OUT
    MOVLW 0FFH
    MOVWF F13
    MOVLW 0FCH
    MOVWF F14

MAIN1
    CALL KEYON     ;CONTROLLO SE CHIAVE INSERITA
    MOVWF F15
    BTFSS F15,0
    GOTO CONFR
    BTFSC F6,1     ;CONTROLLO SE RICHIESTA
                   PROGRAMMAZIONE
    GOTO MAIN1
    BTFSC F5,3     ;CONTROLLO SE CHIAVE ACCESA
    GOTO MAIN1
    CALL DELAY     ;RITARDO PRESSIONE TASTO
    BTFSS F6,1
    GOTO PROG
    GOTO MAIN1


    DATA " CHIAVE ELETTRONICA A 14 INGRESSI "
    DATA " BY Andrea Sbrana "

    ORG 01FFH
    GOTO MAIN
```

*Figura 3. Programma utilizzato nel progetto della chiave elettronica*

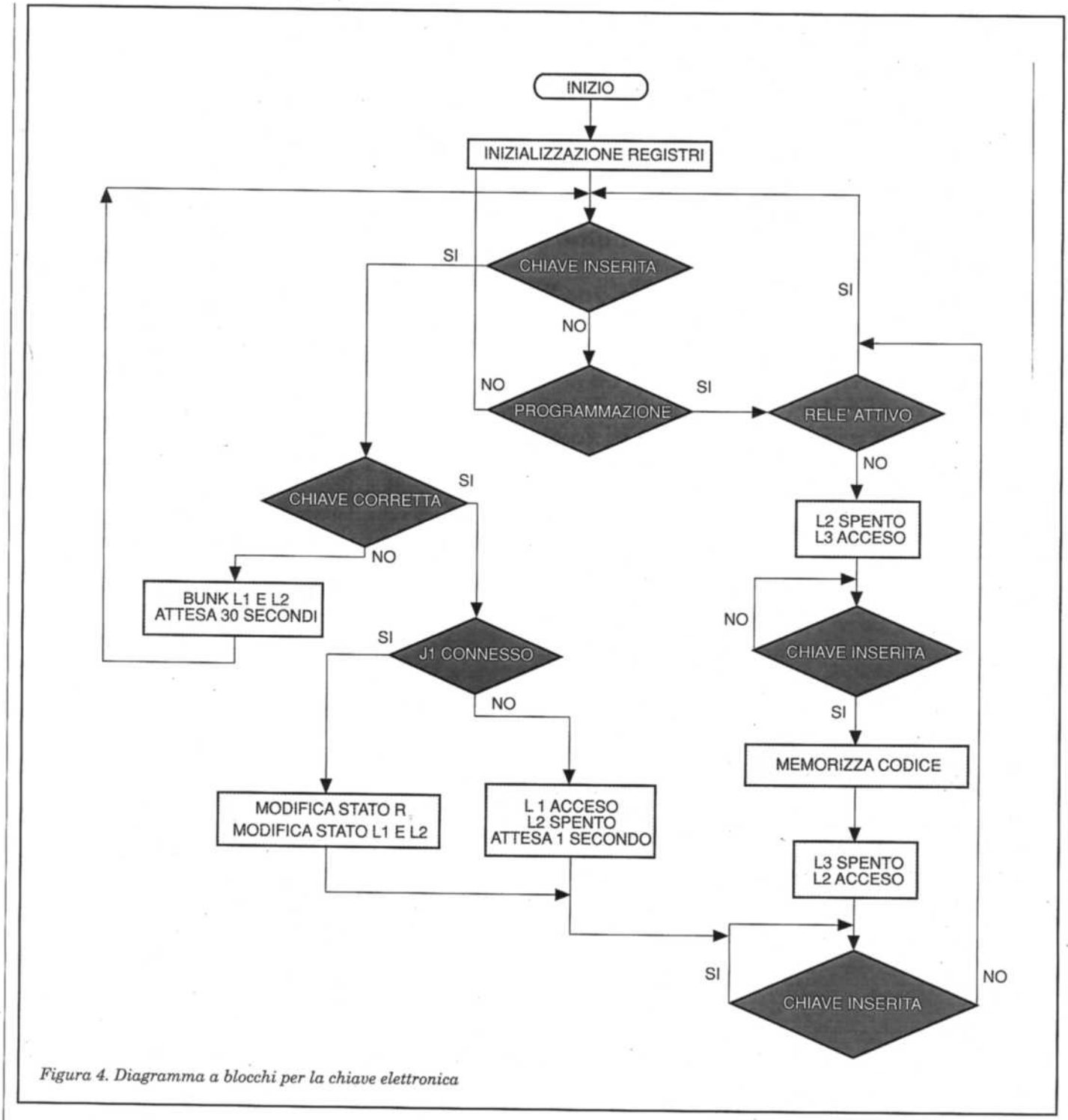



*Figura 4. Diagramma a blocchi per la chiave elettronica*

F15 con il valore ritornato da W e si testa per vedere se è 0 o 1.

Se la chiave risulta inserita, viene invocata la routine CONFR che risponde se è quella corretta oppure no.

Se non è quella corretta, viene introdotto un timer che blocca il PIC per circa trenta secondi, altrimenti si passa alla sezione che controlla le I/O il relè e i relativi Led.

Come esercizio di comprensione delle procedure di programmazione del PIC, potete provare a capire come funziona tale sezione.

Al ritorno dalla CONFR, si testa il bit 1 della porta B (F6) per vedere se è richiesta la programmazione e in tal caso si passa alla PROG.

Infine, si ritorna a MAIN1 e si ricomincia il ciclo.

Il prossimo mese proporremo un metodo per analizzare un programma istruzione peristruzione assistiti da personal computer.

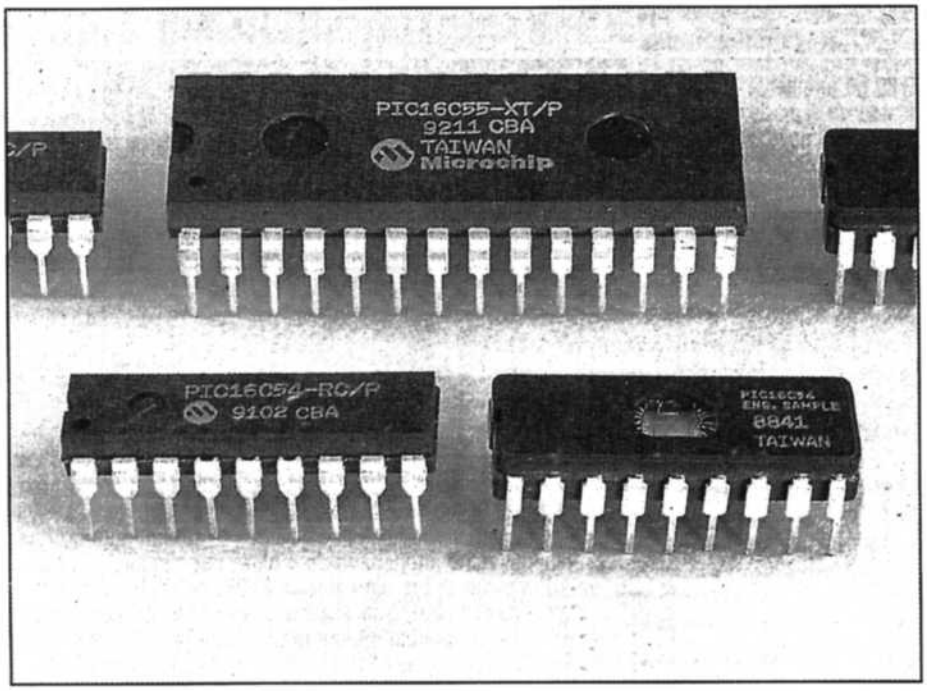
Il consiglio che possiamo darvi è di incominciare a scrivere delle semplici procedure o, addirittura, dei programmi.

■

# PIC16C5x: IMPARIAMO A PROGRAMMARLO

***Nelle precedenti puntate, abbiamo analizzato le caratteristiche e le principali istruzioni dei PIC; è finalmente arrivato il momento di passare "al concreto" e di imparare a usare un assemblatore dedicato.***

*di Andrea Sbrana IW5CBO - 4ª parte*




Dopo aver visto quali sono le istruzioni del microcontroller PIC16C5x, dobbiamo capire con quale particolare attenzione scriverle con un "text editor" su personal computer per ottenere poi un file oggetto, del tipo di quelli presentati fino a ora per la programmazione, e un file "list" che permetterà, in un secondo tempo, di effettuare un comodo e rapido debug del programma stesso per mezzo di un software di simulazione.

Per ottenere questi due file dal sorgente, sfruttiamo un assemblatore realizzato appositamente dalla Hardlab per la famiglia PIC16C5x, quindi molto veloce e sicuro perché dedicato a questo ben preciso impiego; esistono, infatti, assemblatori universali, ma non sempre riescono a coprire tutte le richieste di un determinato controller e in più vanno appositamente programmati per la famiglia desiderata.

L'assemblatore che proponiamo, come già detto, viene prodotto dalla Hardlab e si chiama PIC_ASM.

Per coloro che non dispongono di solide basi di informatica (ovviamente niente di strano per un "elettronico"!) spieghiamo in breve quali sono i compiti che normalmente svolge un assemblatore.

Innanzitutto deve controllare la sintassi del programma sorgente e segnalare tutti gli errori che individua: ad esempio, se scriviamo nel sorgente la riga:

MOVF 05 aa bb ;commento

l'assemblatore dovrà avvisare che la sintassi non è corretta.

Altro compito dell'assemblatore è quello di convertire gli indirizzi relativi ai dati dal sorgente in indirizzi assoluti necessari poi al PIC.

Con il calcolo degli indirizzi dovrà, inoltre, segnalare se questi vanno oltre i limiti della memoria interna del PIC: ad esempio per il PIC 16C54, la memoria interna arriva a 1FFh, mentre per il PIC 16C57 arriva a 7FFh.

L'assemblatore assegna, quindi, a ogni label un indirizzo assoluto che utilizzerà in un secondo momento per rilocare istruzioni di tipò GOTO o CALL o RETLW.

Per ultimo, il PIC-ASM "assembla" le varie istruzioni, cioè a ognuna di esse associa un codice binario che serve al PIC per risalire all'istruzione stessa.

Quello che segue é l'elenco, ovviamente in forma abbreviata e sintetica, dei compiti principali svolti dal nostro compilatore.

## Preparazione di un file sorgente

Vediamo allora come preparare un file sorgente per il nostro assemblatore, esaminando intanto la sintassi del PIC-ASM.

### SET CARATTERI

Ogni assemblatore è in grado di riconoscere un limitato set di caratteri; il nostro riconosce solamente i seguenti:

```
   TITLE   'Lampeggio di un Led'

   LIST P=16C54   ;device scelto

SET_P_A EQU 0FFH
SET_P_B EQU 00FH
TIM0    EQU 79H    ;50 mS  =  79H
TIM1    EQU 10H    ;1 SECONDO  =  10H
F25     EQU 19H    ;RITARDO 1 SECONDO
F26     EQU 1AH    ;RITARDO 1 SECONDO
F27     EQU 1BH    ;RITARDO 1 SECONDO

       ORG 0

       OPTION


;ROUTINE DI RITARDO 1 S
DELAY1
       MOVLW TIM1
       MOVWF F25
DEL2S
       MOVLW TIM0
       MOVWF F26
DEL2A
       MOVWF F27
DEL2B
       NOP
       NOP
       NOP
       DECFSZ F27
       GOTO DEL2B
       DECFSZ F26
       GOTO DEL2A
       DECFSZ F25
       GOTO DEL2S
       RETLW 0


MAIN
   MOVLW SET_P_B
   TRIS PORT_B  ;B0..B3 IN  B4..B7 OUT
   MOVLW SET_P_A
   MOVWF PORT_B
   CLRW         ;azzera W
   TRIS PORT_A  ;A OUT
   MOVWF PORT_A ;Tutti i pin di A a zero

MAIN1
   BTFSC PORT_A,0 ;testa porta A bit 0, skippa se
zero
   GOTO SETTA   ;vai a setta
   BSF PORT_A,0 ;setta a 1 bit 0 di porta A
   CALL DELAY1  ;ritardo di un secondo
   GOTO MAIN1   ;torna a main1

SETTA
   BCF PORT_A,0 ;setta a 0 bit 0 di porta A
   CALL DELAY1  ;ritardo di un secondo
   GOTO MAIN1   ;vai a main1

   DATA "Progetto-corso PIC"

   END
```

*Figura 1. Programma per il lampeggio di un Led*

- ALFABETICI: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
- NUMERICI: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
- SPECIALI: $, &, !, ?, =, £, _, -
- RISERVATI: tab space " ;

Il PIC_ASM non fa distinzione tra caratteri minuscoli e maiuscoli.

Una linea del file sorgente deve rispettare la seguente sintassi:

[<label>] <comando> [<op1>] [<op2>] [<comando>]

Le parentesi quadre indicano che il loro contenuto può essere omesso.

***<label>*** Nome simbolico variabile da due ad otto caratteri.

Non può essere identico ad alcuna delle parole chiave (vedi cap 3,4).

Utile per definire costanti e nomi simbolici all'interno del programma.

***<comando>*** Nome della direttiva assembler oppure dell'istruzione.

***<op1>*** Primo operando per l'istruzione o la direttiva assembler che lo precede. Può essere una label definita come EQU.

***<op2>*** Secondo operando per l'istruzione o la direttiva assembler che lo precede. Può essere una label definita come EQU.

***<commento>*** Eventuale commento utile per il programmatore.

Deve necessariamente iniziare con il carattere ;

È, inoltre, possibile inserire righe solo di commento, senza label o operazioni.

Nota: Le varie voci possono essere suddivise da:

VIRGOLA, SPAZIO, TABULATORE

## COSTANTI

Le costanti, per definizione, devono essere definite sempre maggiori di zero, pena un errore di compilazione.

Possono essere definite in base decimale (il numero stesso) oppure in base esadecimale (il numero esadecimale seguito da H).

Ad esempio: 23 viene interpretato come 17H, mentre 23H viene interpretato come 23H.

Il range di una costante varia da 0 a ffH se tale costante indirizza un registro oppure è l'operando di una istruzione.

Se, invece, la costante rappresenta un indirizzo fisico, potrà variare tra:

- 0..1ff per il PIC 16C54 e PIC 16C55
- 0..3ff per il PIC 16C56
- 0..7ff per il PIC 16C57

## NOMI SIMBOLICI

I nomi simbolici vengono impiegati per definire costanti o per definire LABEL di riferimento.

Un nome simbolico deve essere composto da un numero di caratteri variabile da 2 ad 8.

Se un nome simbolico viene ripetuto

in una definizione, si genera un errore di compilazione.

Alcuni nomi simbolici sono già stati definiti all'interno dell'assemblatore e sono:

## Direttive dell'assemblatore

**TITLE**

SINTASSI: TITLE <nome programma> [<commento>]

DESCRIZIONE: La direttiva TITLE deve essere nella prima riga del programma sorgente e indica il nome del programma stesso.

Non può essere omessa.

```
W = 0           F = 1
PC = 2          STATUS = 3
PORT_B = 6      PORT_C = 7
Z = 2           PD = 3
PA1 = 6         PA2 = 7
PS2 = 2         PSA = 3

F0 = 0          RTCC = 1
FSR = 4         PORT_A = 5
C = 0           DC = 1
TO = 4          PA0 = 5
PS0 = 0         PS1 = 1
RTE = 4         RTS = 5
```

**LIST**

SINTASSI: LIST P=<device> [<commento>]

DESCRIZIONE: La direttiva LIST deve essere nella seconda riga del programma sorgente e indica all'assemblatore il tipo di device che si intende usare. Non può essere omessa.

**EQU**

SINTASSI: <label> EQU <costante> [<commento>]

DESCRIZIONE: EQU permette di associare delle costanti numeriche a dei simboli richiamabili,in un secondo momento, da programma.

**ORG**

SINTASSI: [<label>] ORG <costante> [<commento>]

DESCRIZIONE: Ci deve essere almeno una direttiva ORG prima di una qualsiasi istruzione del PIC.

Generalmente si parte dall'indirizzo 0. La direttiva ORG va usata per indirizzare le pagine di memoria dei PIC 16C56 e 16C57.

```
-Name of file not DOS type
Il nome del file sorgente indicato non è di tipo MSDOS.
-Filename not present
Mancante il nome del file sorgente.
-TITLE line expected
Mancante la direttiva TITLE
-character ' ; ' expected for comment
Il commento deve iniziare con il punto e virgola.
-Parameter expected
Mancanti uno o più parametri.
-MAIN label not present
Mancante la label MAIN.
-LIST line expected
Mancante la direttiva LIST.
-File not found
Il file sorgente non è stato trovato.
-ORG error
La direttiva ORG non è stata ben definita.
-END expected
Mancante la direttiva END.
-Use improper of keyword
Tentativo di usare una label o un simbolo dichiarate
come una parola chiave.
-Call or Retlw overflow
È stata fatta una chiamata ad una routine che non risiede nella
parte alta di una pagina (00..ffH) o che dista per più
 di ffH posizioni.
-Duplicate symbol
Tentativo di ridefinire una label o un simbolo.
-Value incorrect
Il numero indicato non rientra nei parametri consentiti
dall'implementazione.
-Name max 8 character long
Superato il limite massimo di 8 caratteri per una label o un
simbolo.
-Illegal word
Il campo di una direttiva DATA è nullo.
-Unrecognized error
Errore non ben identificato
```

*Figura 2. I principali errori che potranno essere visualizzati dal compilatore*

**DATA**

SINTASSI: DATA <"string"> [<commento>] DESCRIZIONE: DATA consente di inserire da un certo indirizzo in poi dei caratteri alfanumerici contenuti tra doppi apici.

**END**

SINTASSI: END [<commento>]

DESCRIZIONE: La direttiva END determina la fine del programma sorgente. Scrivere END equivale al codice: GOTO MAIN

Infatti, l'assemblatore codifica END come GOTO MAIN, ma in più definisce la fine del programma sorgente.

La label MAIN deve necessariamente essere presente nella prima pagina della memoria.

Con queste informazioni, possiamo allora scrivere correttamente il programma già accennato la scorsa volta per il lampeggio di un Led e visibile in Figura 1.

Per assemblare un file sorgente è sufficiente dare il comando:

CO <nome_file>

dove <nome_file> è il nome MSDOS del file da assemblare.

Per convenzione, tale nome è lungo al massimo 8 caratteri e ha estensione ".ASM"

Se durante la compilazione non vengono riscontrati errori, sul video verranno visualizzati dei punti (.) in corrispondenza di ogni istruzione disassemblata e, al

```
07/03/93   Hardlab inc.PIC CROSS ASSEMBLER v.2.01 page 1
21:32:47        SOURCE FILE: pic4.asm

0001     TITLE   'Lampeggio di un led'
0002
0003     LIST P=16C54   ;device scelto
0004
0005      0FF     SET_P_A    EQU 0FFH
0006      00F     SET_P_B    EQU 00FH
0007      079     TIM0EQU 79H     ;50 mS  =  79H
0008      010     TIM1EQU 10H     ;1 SECONDO  =  10H
0009      019     F25 EQU 19H     ;RITARDO 1 SECONDO
0010      01A     F26 EQU 1AH     ;RITARDO 1 SECONDO
0011      01B     F27 EQU 1BH     ;RITARDO 1 SECONDO
0012
0013                      ORG 0
0014
0015 000 002              OPTION
0016
0017
0018              ;ROUTINE DI RITARDO 1 S
0019              DELAY1
0020 001 C10              MOVLW TIM1
0021 002 039              MOVWF F25
0022              DEL2S
0023 003 C79              MOVLW TIM0
0024 004 03A              MOVWF F26
0025              DEL2A
0026 005 03B              MOVWF F27
0027              DEL2B
0028 006 000              NOP
0029 007 000              NOP
0030 008 000              NOP
0031 009 2FB              DECFSZ F27
0032 00A A06              GOTO DEL2B
0033 00B 2FA              DECFSZ F26
0034 00C A05              GOTO DEL2A
0035 00D 2F9              DECFSZ F25
0036 00E A03              GOTO DEL2S
0037 00F 800              RETLW 0
0038
0039
0040                MAIN
0041 010 C0F    MOVLW SET_P_B
0042 011 006    TRIS PORT_B  ;B0..B3 IN  B4..B7 OUT
0043 012 CFF    MOVLW SET_P_A
0044 013 026    MOVWF PORT_B
0045 014 040    CLRW         ;azzera W
0046 015 005    TRIS PORT_A  ;A OUT
0047 016 025    MOVWF PORT_A ;Tutti i pin di A a zero
0048
0049              MAIN1
0050 017 605    BTFSC PORT_A,0  ;testa porta A bit 0,
                                          ;skippa se zero

07/03/93   Hardlab inc.PIC CROSS ASSEMBLER v.2.01 page 2
21:32:48        SOURCE FILE: pic4.asm

0051 018 A1C    GOTO SETTA   ;vai a setta
0052 019 505    BSF PORT_A,0 ;setta a 1 bit 0 di
                                          ;porta A
0053 01A 901    CALL DELAY1  ;ritardo di un secondo
0054 01B A17    GOTO MAIN1   ;torna a main1
0055
0056              SETTA
0057 01C 405    BCF PORT_A,0 ;setta a 0 bit 0 di
                                          ;porta A
0058 01D 901    CALL DELAY1  ;ritardo di un secondo
0059 01E A17    GOTO MAIN1   ;vai a main1
0060
0061 01F 050    DATA "Progetto-corso PIC"
0062
0063 1FF A10              END  (GOTO MAIN)

07/03/93   Hardlab inc.PIC CROSS ASSEMBLER v.2.01 page 3
21:32:48        SOURCE FILE: pic4.asm

NAME        VALUE  LINE   REFERENCE for EQUATES or CONSTANT

W           000    0000
F           001    0000
F0          000    0000
RTCC        001    0000
PC          002    0000
STATUS      003    0000
FSR         004    0000
PORT_A      005    0000   0046 0047 0050 0052 0057
PORT_B      006    0000   0042 0044
PORT_C      007    0000
C           000    0000
DC          001    0000
Z           002    0000
PD          003    0000
TO          004    0000
PA0         005    0000
PA1         006    0000
C           007    0000
PS0         000    0000
PS1         001    0000
PS2         002    0000
PSA         003    0000
RTE         004    0000
RTS         005    0000
SET_P_A     0FF    0005   0043
SET_P_B     00F    0006   0041
TIM0        079    0007   0023
TIM1        010    0008   0020
F25         019    0009   0021 0035
F26         01A    0010   0024 0033
F27         01B    0011   0026 0031
07/03/93   Hardlab inc.PIC CROSS ASSEMBLER v.2.01  page
21:32:48        SOURCE FILE: pic4.asm

NAME        VALUE  LINE   REFERENCE for LABEL

DELAY1      001    0019   0053 0058
DEL2S       003    0022   0036
DEL2A       005    0025   0034
DEL2B       006    0027   0032
MAIN        010    0040
MAIN1       017    0049   0054 0059
SETTA       01C    0056   0051
```

*Figura 3. Il file PIC4.LS*

termine, compare la sospirata e spesso tanto attesa frase:

Compiled File Stored

Se, invece, saranno rilevati errori, l'assemblatore visualizzerà un messaggio di errore: nella Figura 2, sono elencati gli errori principali.

Parallelamente all'assemblaggio del file PIC4.ASM (il nome assegnato al nostro file di prova), se non vengono riscontrati errori, sono prodotti i due file PIC4.LST e PIC4.OBJ: cerchiamo di capire come sono fatti e come impiegarli per una corretta programmazione e per un veloce debug.

Nella Figura 3 potete vedere il file PIC4.LST.


***" I PIC sono i controller del futuro ! PROGETTO vi dà la possibilità di anticipare i tempi..."***

Come potete notare, la struttura generale di un file LST è composta generalmente da:

- Prima colonna: Numero di linea del file sorgente
- Seconda colonna: Indirizzo relativo al program counter del PIC
- Terza colonna: Istruzione disassemblata
- Successive: Dati letti sul file sorgente

Inoltre, in questo file sono presenti due sezioni, una dedicata ai riferimenti della direttiva EQU, l'altra per i riferimenti delle label trovate. Per i riferimenti EQU abbiamo la seguente struttura:

- Prima colonna: Nome del simbolo
- Seconda colonna: Valore del simbolo rappresentato in esadecimale
- Terza colonna: Numero della linea in cui tale simbolo è stato definito
- Quarta colonna e successive: Numero delle linee dove vengono richiamati i simboli

Per i riferimenti alle label, invece, abbiamo:

- Prima colonna: Nome della label
- Seconda colonna: Valore della label rappresentato in esadecimale
- Terza colonna: Numero della linea in cui tale label è stata definita
- Quarta colonna e successive: Numero delle linee dove vengono richiamate le label.

In Figura 4, invece, è visibile il file PIC4.OBJ.

Il formato di uscita è detto comunemente 16-Bit Word.

Ogni record dati inizia con un prefisso di 9 caratteri e termina con un checksum di 2. Ogni record ha il seguente formato:

:BBAAAATTHHHH....HHHHCC
dove:

: è il carattere iniziale.

BB è il numero dei dati inseriti nel record (max 8).

AAAA è l'indirizzo iniziale per le word nel record.

TT è il tipo del record (vale sempre 00 eccetto che per il record finale che è 01).

HHHH è una word di dato (espressa in esadecimale).

CC è il checksum di tutto il record.

Il file PIC4.OBJ, è poi quello che sarà impiegato dal programmatore hardware vero e proprio. Tornando al PIC4.LST, notiamo che è possibile seguire passo passo ogni istruzione, ogni riferimento alle EQU e alle label, cosa molto utile per "debuggare" il programma nel caso in cui non si comporti come dovrebbe.

Il prossimo mese, infatti, mostreremo un software di simulazione su personal computer che, passo-passo, unitamente al file LST e al file OBJ, simulerà tutta la famiglia dei PIC16C5x, cioè consentirà di "vedere" che cosa succede internamente ai PIC dopo ogni istruzione, sia a livello dei registri, sia a livello delle porte di input-output.

- contiua -

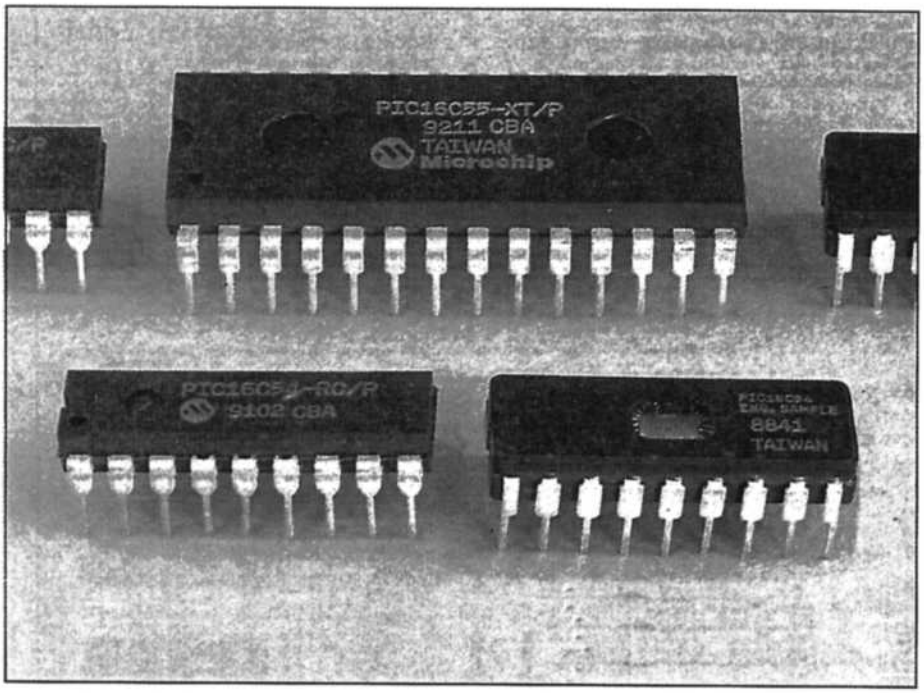
```
:0800000000020C1000390C79003A003B00000000A7
:08000800000002FB0A0602FA0A0502F90A030800C8
:080010000C0F00060CFF00260040000050025060521
:080018000A1C050509010A17040509010A17005001
:08002000072006F006700650074007400 6F002DA7
:08002800006300 6F00720073006F002000500049F1
:0100300000438C
:0101FF000A10E5
:00000001FF
```

*Figura 4. Il file PIC4.OBJ*

# PIC16C5x: IMPARIAMO A PROGRAMMARLO

***Una volta analizzato l'assemblatore e le istruzioni del PIC, siamo pronti a iniziare la descrizione del software che gestisce sia la parte di simulazione che quella di programmazione della famiglia PIC16C5x.***

*di Andrea Sbrana IW5CBO - 5ª parte*




Come per l'assemblatore descritto nella scorsa puntata, il PIC-Programmer viene prodotto dalla Hardlab e offre le seguenti funzioni principali:

- Programmazione dei PIC (unitamente al relativo hardware);
- Caricamento in RAM del PC del file OBJ prodotto dal PIC-ASM con possibilità di editor in linea;
- Verifica della corretta programmazione dei PIC;
- Lettura del software memorizzato su PIC (sempre che questo non sia protetto).

In Figura 1 potete vedere la prima schermata che si presenta non appena il software viene attivato con il comando EM.

Questo comando, al momento della stesura definitiva del programmatore, potrà subire qualche variazione che ovviamente sarà inclusa in un file di tipo READ.ME.

La prima riga partendo dall'alto indica il nome della ditta produttrice (Hardlab), la descrizione del software (PIC EMULATOR/PROGRAMMER), la data espressa in giorno/mese/anno e l'ora espressa in ore/minuti.

Ci sono poi una serie di righe che indicano le varie opzioni possibili con la lettera iniziale di colore diverso dalle altre: premendo il tasto relativo a una lettera verrà attivata la routine corrispondente.

In fondo troviamo dei settaggi da effettuare non appena lanciato il programma: con (F1) si può variare il tipo di device (esempio PIC 16C54, 16C55 ecc), con (F2) il tipo di oscillatore (esempio RC o XT), con (F3) si indica, invece, se vogliamo proteggere dalla lettura i dati inseriti nel PIC (utili per applicazioni industriali e commerciali), con (F4) segnaliamo se deve essere attivato il watchdog (cane da guardia) interno che resetta il PIC ogni 18 mS se non azzerato prima durante l'esecuzione del programma. Infine, con (F7) è possibile selezionare la porta seriale 1 o la 2 (COM1 o COM2).

L'ultima riga visualizza i messaggi di sistema utili per l'utente, come per esempio un file non trovato o il programmatore non presente.

Riprendiamo allora il programma analizzato lo scorso mese per il lampeggio di un LED e settiamo correttamente il programmatore: il device deve essere di tipo 16C54, l'oscillatore di tipo RC, la protezione per ora è meglio non inserirla (NOT), il watchdog deve essere non attivo (NOT) e, infine, la seriale va settata in funzione di quella da voi utilizzata per il programmatore (che presenteremo il prossimo mese).

A questo punto, nella stessa directory del software di programmazione e di emulazione, dovrete avere il file LAMP.OBJ (o con altro nome, ma sempre con estensione OBJ) assemblato lo scorso mese. Caricatelo premendo il tasto L e scrivendo il nome del file seguito da <Invio>.

Apparirà allora un messaggio inerente lo stato dell'istruzione data: se il file non viene trovato sarà segnalato con il messaggio OBJECT FILE NOT FOUND, se viene trovato, ma non è del tipo richiesto oppure contiene alcuni

errori apparirà il messaggio WRONG FILE: ERROR IN RECORD xx. Viceversa il file sarà caricato e apparirà il messaggio OBJECT FILE LOADED.

Con il file caricato possiamo fare di tutto e, per prima cosa vediamo come gestire l'editore di testi a disposizione: premete allora il tasto E.

La prima schermata sarà così sostituita da quella visibile nella Figura 2 e che rappresenta la pagina dell'editore del PIC-Programmer.

La prima riga è identica alla schermata iniziale. Le successive indicano sull'estrema sinistra l'indirizzo iniziale della stringa di operazioni immediatamente a destra. Ad esempio la riga:

000 068 C01 088 000 CFF 029 040 089

indica che l'istruzione 068 è all'indirizzo 000, l'istruzione CFF è all'indirizzo 004 e così via. Ricordiamo che l'indirizzo di cui parliamo è proprio quello della memoria fisica dentro il PIC.

Sulla parte destra dello schermo troviamo, invece, l'istruzione puntata attualmente dal cursore e si ha notizia dell'indirizzo, del codice dell'istruzione e perfino del disassemblaggio stesso dell'istruzione (ad esempio per l'istruzione 029 viene indicato MOVWF F09).

In basso a destra troviamo tre comandi da dare: (F1) permette di eliminare un'istruzione, (F2) di inserirne una in più, (F5) di rimemorizzare il nuovo file. Da notare che questi passaggi non sono poi così banali, poiché il software deve tener conto di tutti gli indirizzamenti, dei salti condizionati e non e deve poi ricostruire il file OBJ con il cecksum finale per ogni record.

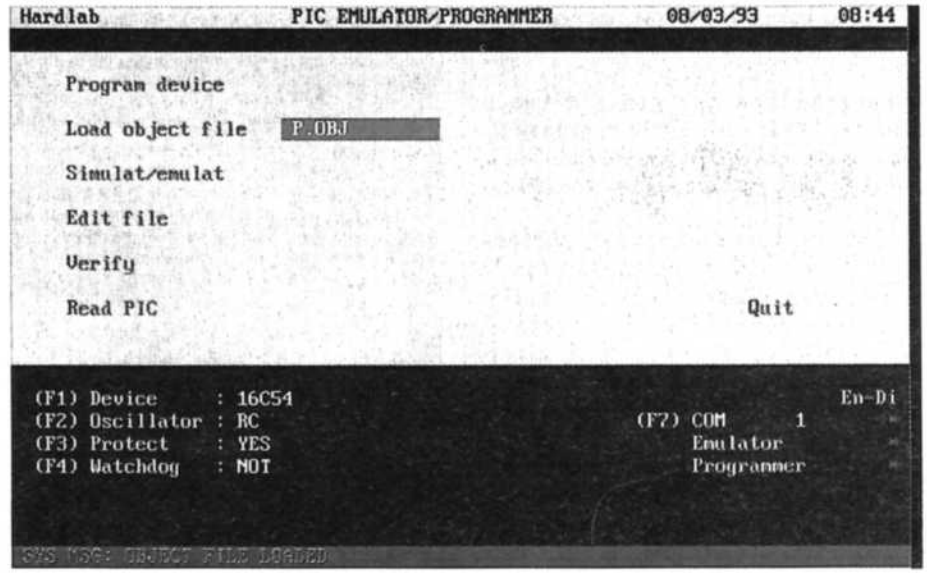



*Figura 1.Schermata iniziale del programmatore per PIC*

I tasti che possono essere utilizzati sono Page-Up e Page-Down per indirizzare tutti gli indirizzi possibili di un PIC, i tasti cursore per spostare il cursore nella zona dati, i numeri da 0 a 9 e le lettere da A a F per modificare le istruzioni e, infine, il tasto ESCAPE per tornare alla maschera iniziale.

Poter lavorare con un editore con queste caratteristiche è utilissimo durante le fasi di simulazione del PIC.

La simulazione del PIC, infatti, rappresenta uno dei migliori sistemi per verificare se un software riproduce esattamente le funzioni che noi desideriamo oppure no.

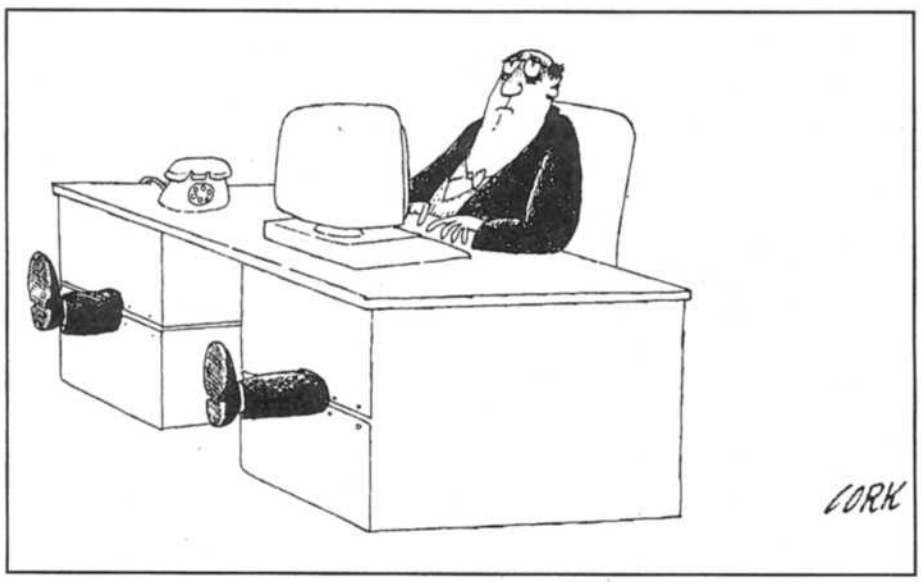
Per entrare in ambiente di simulazione, basta premere il tasto S dal menu iniziale.

In Figura 3 troviamo la maschera relativa a questo ambiente: tutti i registri del PIC vengono visualizzati in esadecimale e, i più importanti, anche in binario.

Al centro vediamo l'istruzione appena eseguita e, sotto, quella successiva già "puntata" dal program counter per effetto del ciclo di fetch.

Con i tasti funzione esiste la possibilità di:

- Modificare il contenuto di tutti i registri, sia in modo binario che esadecimale (F1);
- Variare la frequenza di lavoro del PIC (F2);
- Far eseguire un certo numero di cicli al simulatore per poi visualizzare solo il risultato finale onde guadagnare tempo (F3);
- Inserire un breakpoint a un determinato indirizzo (F4);
- Resettare il tempo di esecuzione parziale per poter ad esempio calcolare il ritardo di una routine (F5);
- Resettare completamente l'emulatore per una nuova simulazione (F6);
- Memorizzare l'intera configurazione del simulatore fino a un certo momento, molto utile per non ripetere tutti i passaggi già fatti e collaudati (F7);
- Richiamare la configurazione salvata precedentemente (F8);
- Eseguire una istruzione alla volta (passo-passo) con il tasto SPACE;



- Visualizzare tutti i registri del PIC (PgUp, PgDw)..

La penultima riga indica il tempo trascorso dall'inizio della simulazione (in microsecondi), il tempo parziale (sempre in microsecondi) e il numero di istruzioni eseguite.

Provate allora a simulare il programma del lampeggio di un LED e potrete calcolarvi i tempi desiderati semplicemente lanciando la simulazione e attendendo i dati sul video: se i valori letti non corrispondono a quelli attesi, potete o variare i contenuti di alcuni registri all'interno del simulatore, oppure tornare in ambiente editor e modificare direttamente le istruzioni.

Dopo un brevissimo periodo di prova, realizzare un programma funzionante per i PIC sarà estremamente facile e diventerà un divertimento!

Provate a seguire il programma per il lampeggio di un LED descritto nelle puntate precedenti con il simulatore cercando di capire i vari passaggi e facendo attenzione ai contenuti di tutti i registri dopo ogni singola istruzione e alla fine potrete simulare qualsiasi altro programma.

## I messaggi

I messaggi che il sistema può visualizzare sono:

**EDIT SESSION:** Indica l'entrata nella sessione di editing;

**FILE STORED:** Segnala l'avvenuta memorizzazione di un file OBJ modificato;

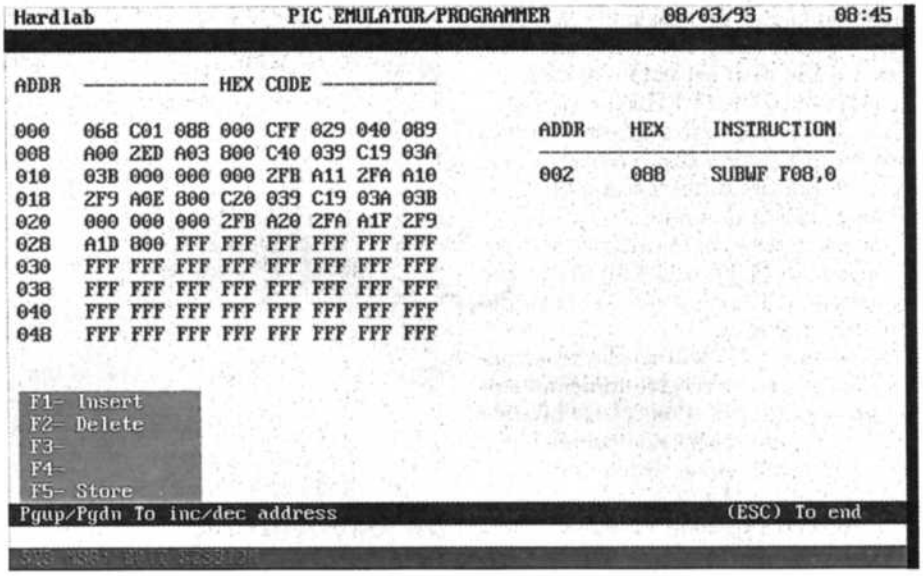



*Figura 2. Schermata dell'editor del programmatore di PIC*

**FILE(s) ON DISK -C- ONLY:** Avvisa che i file possono essere letti e/o scritti solamente dal disco fisso C;

**FILE NOT FOUND:** Il file da caricare non è stato trovato;

**OBJECT FILE NOT LOADED:** Segnala che il comando richiesto non può essere eseguito perché il file OBJ non è stato caricato in memoria;

**OBJECT FILE LOADING:** Lampeggia quando il caricamento di un file OBJ è in corso;

**OBJECT FILE LOADED:** Indica l'avvenuto caricamento di un file OBJ;

**WRONG FILE: ERROR IN RECORD xx:** Segnala che il file OBJ non è corretto e indica in quale record ha riscontrato errore;

**EMULATOR IS RUNNING:** Segnala l'entrata in ambiente simulazione;

**PROGRAMMER NOT PRESENT:** Segnala che non è possibile programmare un PIC perché non è connesso il programmatore;

**SERIAL PORT x NOT READY:** Indica quale porta seriale non è pronta;

**STACK UNDERFLOW:** In ambiente di simulazione avvisa quando è avvenuto un underflow sullo stack (più di due RETLW consecutive);

**STACK OVERFLOW:** In ambiente di simulazione avvisa quando è avvenuto un overflow sullo stack (più di due CALL consecutive);

**DEVICE HAS BEEN PROGRAMMED SUCCESSFULLY:** Segnala che il PIC è stato programmato correttamente;

**PIC VERIFY IS OK:** Dopo una verifica, il PIC è stato trovato OK;

**PIC HAS WRONG DATA:** Dopo una verifica, si sono riscontrati errori nel PIC;

**PIC WAKENED BY WATCHDOG:** In ambiente di simulazione, avvisa che il PIC, dopo un'istruzione SLEEP, è stato svegliato da un reset causato dal watchdog;

**PIC WAKENED BY MCLR:** In ambiente di simulazione, avvisa che il PIC, dopo un'istruzione SLEEP, è stato svegliato da un impulso di reset sul pin MCLR.

- *continua* -

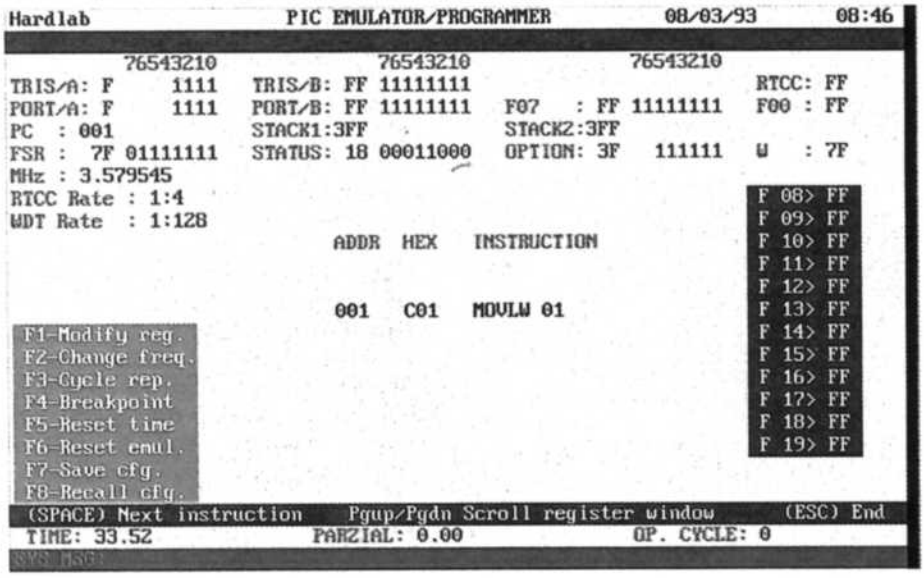



*Figura 3. Visualizzazione dello stato dei registri del PIC*