LUCIDI LEZIONI SISTEMI

ELETTRONICI INDUSTRIALI

www.dismi.unimo.it

(People – Associated – Pavan)

www.microchip.com

(Datasheet PIC - DS30292)

(Mid Range Family Manual – DS33023)

Università degli Studi di Modena e Reggio Emilia

- Facoltà di Ingegneria - sede di Reggio Emilia -

INTRODUZIONE alla PROGRAMMAZIONE di MICROCONTROLLORI

PIC16F877 - Microchip

Procedura di Sviluppo

TUTTI i microcontrollori Microchip utilizzano come ambiente di sviluppo il software **MPLAB**.

Con questo è possibile sviluppare codice in linguaggio C o in linguaggio Assembler.

Una volta generato è possibile scaricare il codice sulla **memoria flash** a bordo del µC utilizzando il **programmatore ICD**.

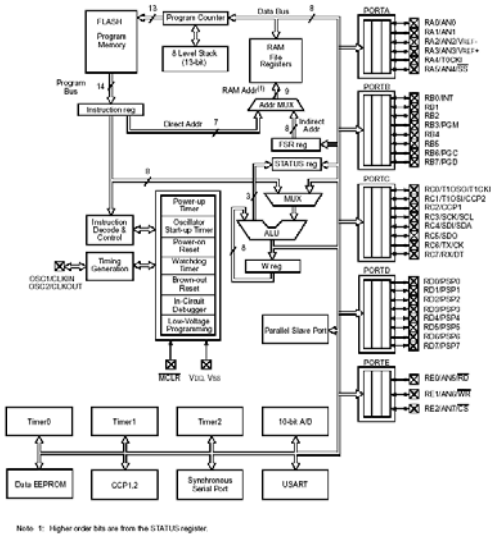
Una volta programmato il microcontrollore è pronto per essere inserito sulla scheda che realizza l'applicazione da controllare.

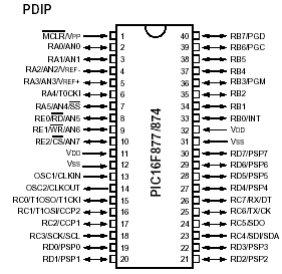
Con il codice generato è possibile "istruire" il microcontrollore sulle funzioni da compiere.

NOI UTILIZZEREMO PIC 16F877 e li programmeremo in Linguaggio Assembler

Architettura Interna del PIC16F877

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes





ALU

W register

I/O Port

Periferiche

- **Timer 0,1,2**
- **ADC 10 bit**
- **SSI & I²C**
- **USART**
- **Parallel Port**
- **PWM**
- **EEProm DATI**

Istruzioni Assembler

Il 16F877 lavora a **8 bit** per cui per variabili che necessitano di più di 1 byte è necessario riservare 2 o più locazioni di memoria.

Il micro esegue **1 istruzione ogni ciclo di clock** (ad eccezione per le istruzioni di goto e call che richiedono 2 cicli).

La frequenza del clock di sistema è pari a ¼ della frequenza imposta dal circuito di temporizzazione (Fosc del quarzo).

La **memoria** è divisa in **4 banchi**, ognuno ha una parte riservata a registri di sistema e una parte di registri general purpose utilizzabili dall'utente (ad esempio per le variabili di lavoro).

Prima di eseguire ciascuna istruzione è buona norma assicurarsi di operare nel banco di memoria giusto:

AGENDO SUI BIT RP1,RP0 dello STATUS REGISTER

UTILIZZANDO LA DIRETTIVA **banksel xxx**

Istruzioni Assembler

Il PIC ha un architettura RISC per cui il numero totale di istruzioni è limitato a circa 35.

Esistono 3 tipi di formato di istruzioni:

- BYTE ORIENTED [Istruz.] f,d
- BIT ORIENTED [Istruz.] f,b
- LITERAL [Istruz.] k

d = 0 = workingREG (W)

d = 1 = stesso REG

Il fatto di inserire i file xxx.inc consente di utilizzare delle macro simboliche che permettono di semplificare la programmazione. Infatti questi file assegnano ad ogni bit di ciascun registro un NOME SIMBOLICO, che è lo stesso di quello riportato sul datasheet.

Es: bcf STATUS, 5 = bcf STATUS, RP0

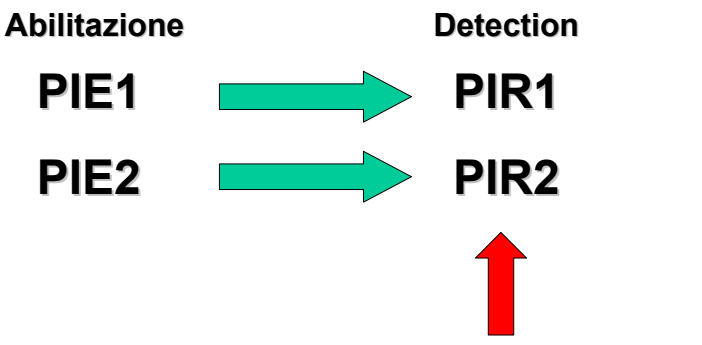
Gestione di Interrupt - Registri

Quando si verifica un Interrupt il PIC blocca immediatamente quello che sta facendo e salta alla locazione di memoria 0x04 a partire dalla quale è allocata l'ISR (Interrupt Service Routine).

I registri base per la gestione di Interrupt sono i seguenti:

Registro	Bit	USO
INTCON	GIE	Abilitazione Generale all' utilizzo di Interrupt
	PEIE	Abilitazione al Riconoscimento di Interrupt provenienti dalle periferiche integrate sul PIC
	INTE	Abilitazione al Riconoscimento di Interrupt provenienti dall' esterno (linea RB0)
PIE1	xxxE	Contengono i flag (bit) corrispondenti alle periferiche di cui interessa rilevare l'interrupt
PIE2	xxxE	
PIR1	xxxF	Contengono i flag (bit) che rilevano il verificarsi di un interrupt generato dalla corrispondente periferica
PIR2	xxxF	

Gestione di Interrupt - Associazione



Struttura Base di un Programma Assembler

```
processor xxxxx
include “xxxxx.inc”
__config xxx  (opzionale)
CBLOCK
        <definizione variabili di lavoro>
ENDC
ORG 0x00
nop
goto xxx
ORG 0x04
        <ISR Interrupt Service Routines>
xxx
<Programma Principale>
        <Inizializzazione Porte & Periferiche>
        <Abilitazione Interrupt>
        <Elaborazione Segnali & Dati>
        call …..
….
<Routines di Calcolo o di Servizio Interrupt invocate>
        …return
END
```

Procedura di Servizio Interrupt e Code Flow

ROUTINE DI SERVIZIO INTERRUPT (A PARTIRE DALL’ INDIRIZZO 0x04)

<Salvataggio Stato del Sistema (Status e W reg.)> (opzionale)
<Ciclo Riconoscimento Interrupt>
btfsc <registrox>, <bit x (flag)>
goto <label xxx1>
btfsc <registroy>, <bit y (flag)>
goto <label xxx2>
…
<Chiamata a Funzioni di Interrupt Service>
xxx1 call <routine1>
goto FISR
(è una label non 1 istruz.!!)
xxx2 call <routine2>
goto Fine_ISR
…
FISR <Ripristino Stato del Sistema>
retfie

FINE ROUTINE DI SERVIZIO INTERRUPT

…

PROGRAMMA PRINCIPALE

…

<Routine di programma>

<Routine di Interrupt>

…

SALTO A ZONA DI CODICE DIVERSA, CHE HA COME LABEL IL NOME DELLA FUNZIONE DA ESEGUIRE

routine1
<Disabilitazione Generale degli Interrupt>
bcf INTCON, GIE
…
<Elaborazione>
<eventuali call ad altre funzioni>
…
<Reset Flag dell’Interrupt servito>
bcf <registrox> , <bit x (flag)>
questo serve per poter riconoscere altri interrupt di questo tipo!!!
<Riabilitazione Interrupt>
bsf INTCON, GIE
return

TERMINATA LA ROUTINE SI TORNA ALL’ISTRUZIONE IMMEDIATAMENTE SEGUENTE ALLA CALL