www.danielepostacchini.it

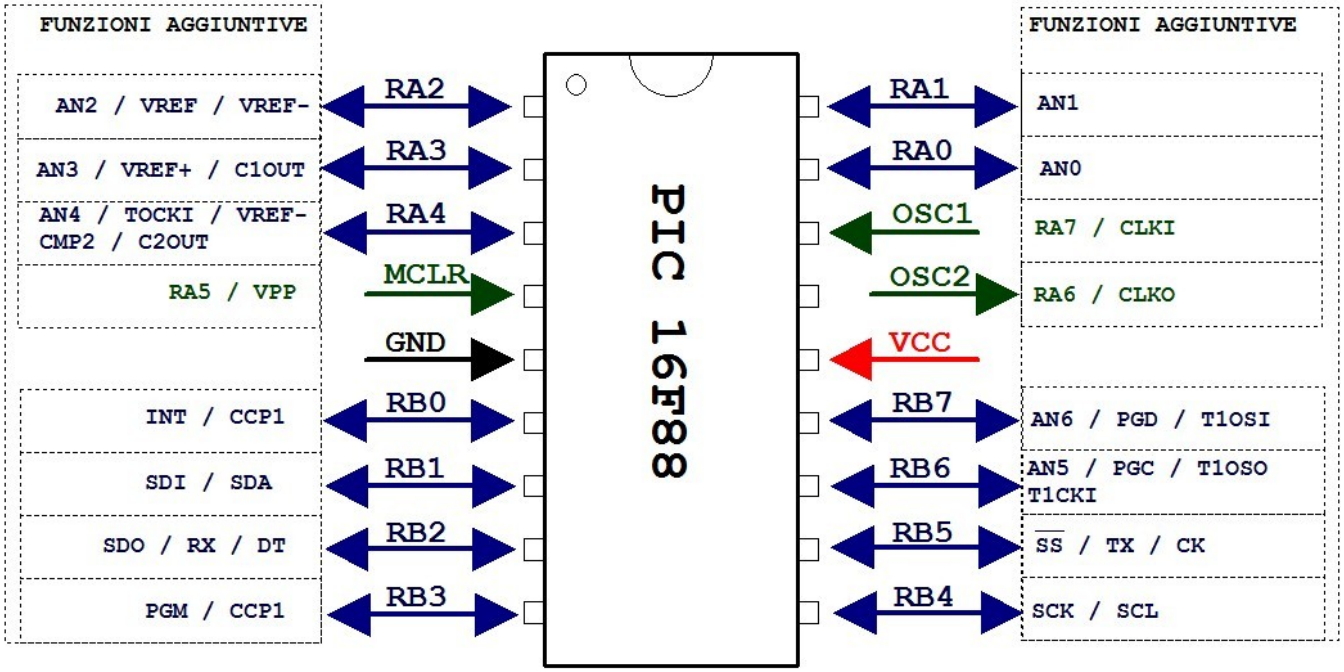
MICROCONTROLLORI MICROCHIP PIC

Un microcontrollore è un componente elettronico che integra in un unico chip tutti i dispositivi per realizzare un sistema completo programmabile, al suo interno possiamo trovare:

- **CPU** — **C**entral **P**rocessing **U**nit ovvero unità centrale di elaborazione.
- **Memoria FLASH** — Una memoria **non volatile** e riscrivibile dove allocare il programma da eseguire.
- **EEPROM** — **E**rasable **El**ectrically **P**rogrammable **R**ead **O**nly **M**emory ovvero una memoria **non volatile** e riscrivibile a cui si può accedere per il salvataggio di dati, utilizzabile dal programma.
- **RAM** — **R**andom **A**ccess **M**emory, ovvero la memoria **volatile** dove salvare le variabili utilizzate. In questa memoria ci sono allocati anche i registri per applicazioni specifiche, come **PORTA**, **PORTB**, **TRISA**, **TRISB**, **STATUS**, ecc...
- **GPIO** — Linee di Input Output (**G**eneral **P**urpose **I**nput **O**utput).
  Nel microcontrollore sono già implementati dei circuiti che consentono di controllare direttamente i piedini del componente, e di configurarli, come digitali o analogici, o come ingresso o uscita.
- **DISPOSITIVI AUSILIARI** — Nel microcontrollore ci sono altri circuiti elettronici già realizzati, quali; timer, contatori, whatchdog (timer che resetta periodicamente il programma), circuiti per la comunicazione seriale, convertitori analogico-digitali ecc....

www.danielepostacchini.it

Un microcontrollore si presenta come un normale circuito integrato, ad esempio il **PIC16F88** si trova anche nel classico contenitore **D**ual **I**n **P**ackage (D.I.P.).



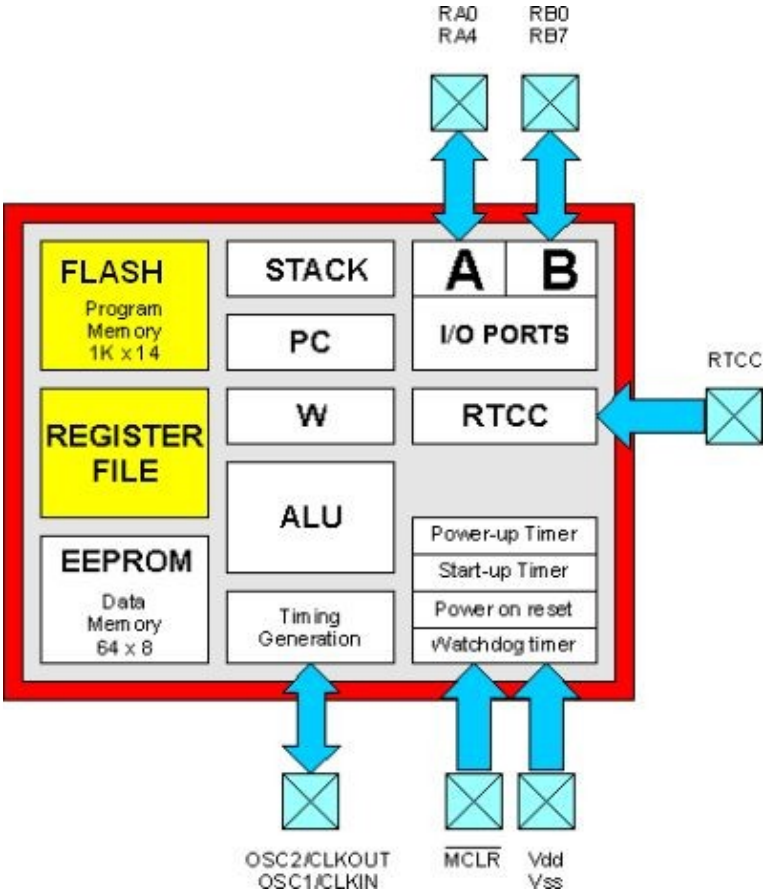
I piedini **BLU** rappresentano le linee di ingresso/uscita **GPIO (General Purpose Input Output)** e possono anche avere più di una funzione. Per scegliere se sono ingressi o uscita o per selezionare altre funzioni aggiuntive, bisogna modificare il contenuto di determinate zone di memoria interne al PIC, chiamate **REGISTRI**.

I piedini **VERDI** si utilizzano per collegare un **CLOCK** esterno o un segnale di reset, anch'essi possono hanno la possibilità di essere utilizzati per altre funzioni o anche come semplici input o output.
Ai piedini chiamati **OSC1** e **OSC2** va collegato un circuito di clock, spesso realizzato con un quarzo.
Il piedino **MCLR** rappresenta il **Master Clear** e cioè il **reset**, in questo caso viene utilizzato anche dal programmatore che caricherà il programma sul microcontrollore. Questa operazione viene fatta mettendo sul piedino **MCLR** un valore di tensione più elevata di quella di alimentazione che è di 5 Volt, operazione che viene svolta autonomamente dal dispositivo programmatore.
Questi piedini possono essere comunque utilizzati per altre funzioni o come semplici input o output.
Il pin **Vdd** rappresenta l'alimentazione di 5Vdc e **Vss** il negativo dell'alimentazione.

www.danielepostacchini.it

STRUTTURA INTERNA DI UN MICROCONTROLLORE

Questa figura rappresenta la struttura interna di un microcontrollore PIC16F84, uno dei microcontrollori della famiglia microchip più utilizzati in passato.



La memoria **FLASH** in alto a sinistra è quella che conterrà il **programma da eseguire.**

La **EEPROM** in basso a destra è quella messa a disposizione all'utente che realizzerà il programma, dove potranno essere memorizzate le **variabili ritentive**, cioè quelle che debbono mantenere il valore anche dopo lo spegnimento.

La **RAM** che contiene le variabili temporanee del programma è rappresentanta dall'area chiamata **Register file**.

Nella zona centrale troviamo lo **STACK** ed il **PC**, che sono due **registri** che non vengono utilizzati direttamente dall'utente, ma che sono gestiti internamente dal microcontrollore per poter leggere e decodificare le righe di programma.

La zona con la scritta **ALU** è quella dove vengono effettuati i calcoli. **A**ritmetic **L**ogic **U**nit **(ALU**).

Al centro troviamo il **registro W (registro accumulatore)**. Esso rappresenta una variabile di memoria da 8 bit dove transitano tutti i dati che vengono elaborati dal programma.

Nella zona di destra, troviamo i dispositivi elettronici interni come ad esempio il **wathcdog**, cioè un timer che se attivato resetta periodicamente il microcontrollore.

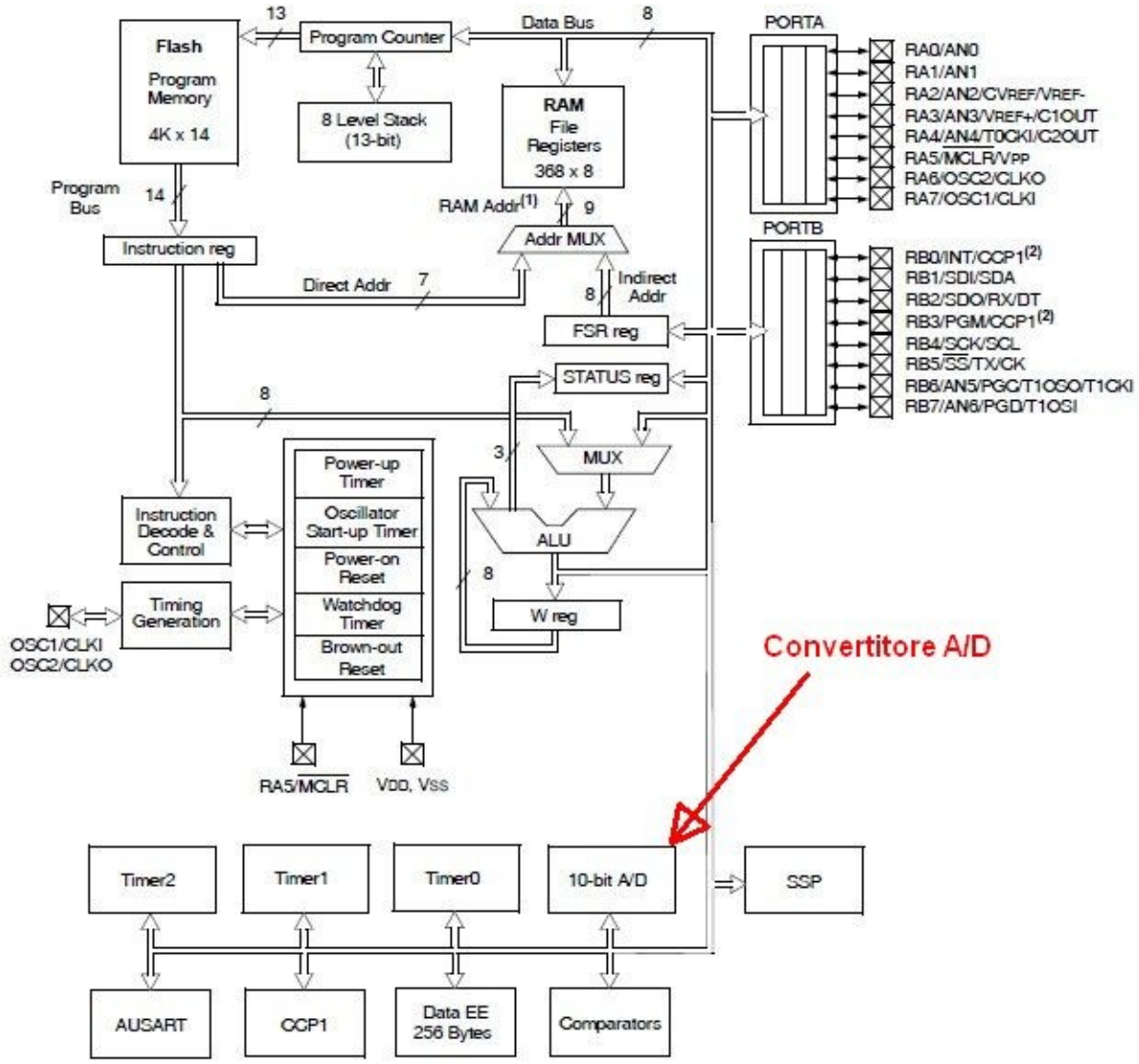
Questo viene utilizzato per evitare che il programma finisca in dei loop imprevisti. Chi scrive il software dovrà ricordarsi all'interno del programma, di reinizializzare il timer watchdog in ogni parte del programma. In questo modo se il watchdog non viene reinizializzato, il timer esaurisce il suo tempo andando a resettare il microcontrollore togliendono perciò dalla situazione di loop.

I piedini di ingresso/uscita denominati con **PORTA** e **PORTB**, assumono il valore dei bit presenti in due omonimi registri a cui il programma può accedere in ogni momento.

www.danielepostacchini.it

Un microcontrollore più evoluto del **PIC16F84**, è il **PIC16F88** che ha una piedinatura compatibile ma ha una maggiore dotazione hardware interna.

La struttura interna del PIC16F88 è infatti più complessa, al suo interno troviamo ad esempio altri circuiti, tra cui un **convertitore Analogico/Digitale a 10 bit** con 7 possibil canali di ingresso (identificati con il nome AN0, AN1....ecc...).



Al centro notiamo inoltre altri circuiti legati ad un generatore di tempi (**Timing Generation)** come ad esempio il **Power-up Timer**, un temporizzatore che ritarda l'avvio del programma al'accensione. Attivando questa funzionalità il microcontrollore comincerà ad eseguire l programma al termine dei transitori della tensione di alimentazione, cioè quando il valore della tensione presente sul pin Vdd è stabile..

Come già detto oltre alle memorie RAM ed EEPROM, nel micorontrollore sono disponibili dei pint GPIO, organizzati secondo una struttura a byte (8 bit) a cui viene assegnato il nome di PORT (inteso come porta di ingresso o uscita) nel nostro caso abbiamo PORT A, e PORT B entrambe ad 8 bit.

Oltre a queste parti possiamo riconoscere nella struttura interna anche l'unita aritmetico logica (**ALU**) , ed il resgistro **accumulatore W** ed il **program counter (PC)**, presenti in ogni microprocessore.

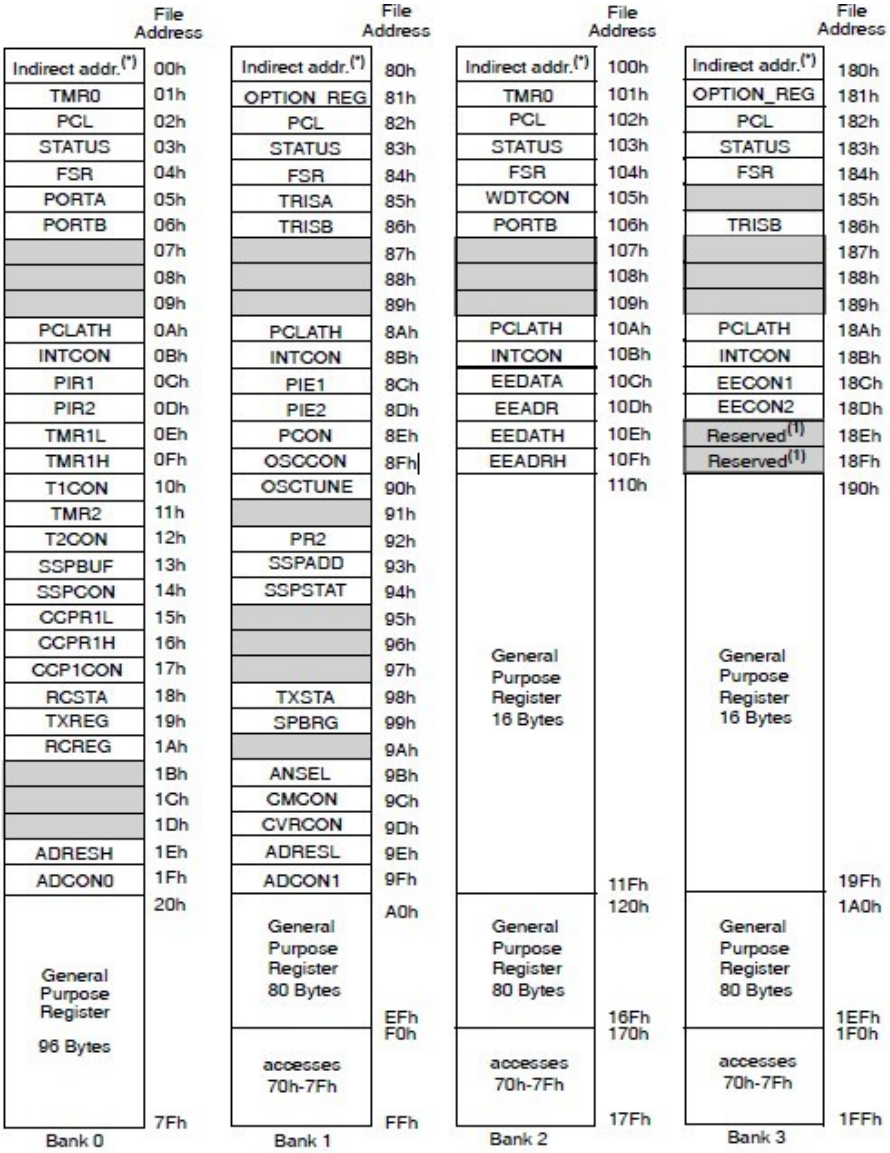
Come si può notare la memoria del programma e la memoria dati sono diffferenti, in questo caso parliamo di architettura **Harvard**, differente dall'architettura **Von Newmann** che condivide per i dati e per il programma lo stesso spazio di memoria.

www.danielepostacchini.it

RAM ED AREA REGISTRI

Atra importante caratteristica del microcontrollore, è la struttura della RAM indicata nello schema a blocchi con **File Register**. Questa memoria ha una struttura predefinita con delle parti chiamate **REGISTRI**, che hanno delle funzioni definite e descritte nel data-sheet del componente. Nella RAM c'è anche una zona che può essere utilizzata liberamente per la creazione di variabili definita **General Purpos Register**.

La struttura della RAM è la seguente:



Come si può notare dall'immagine, la RAM è inoltre divisa in 4 banchi, e per accedere ad uno di questi occorre impostare i due bit (**RP0 ed RP1**) contenuti nel registro **STATUS** che è accessibile in ogni banco di memoria, in modo da attivare la scrittura/lettura sul relativo banco.

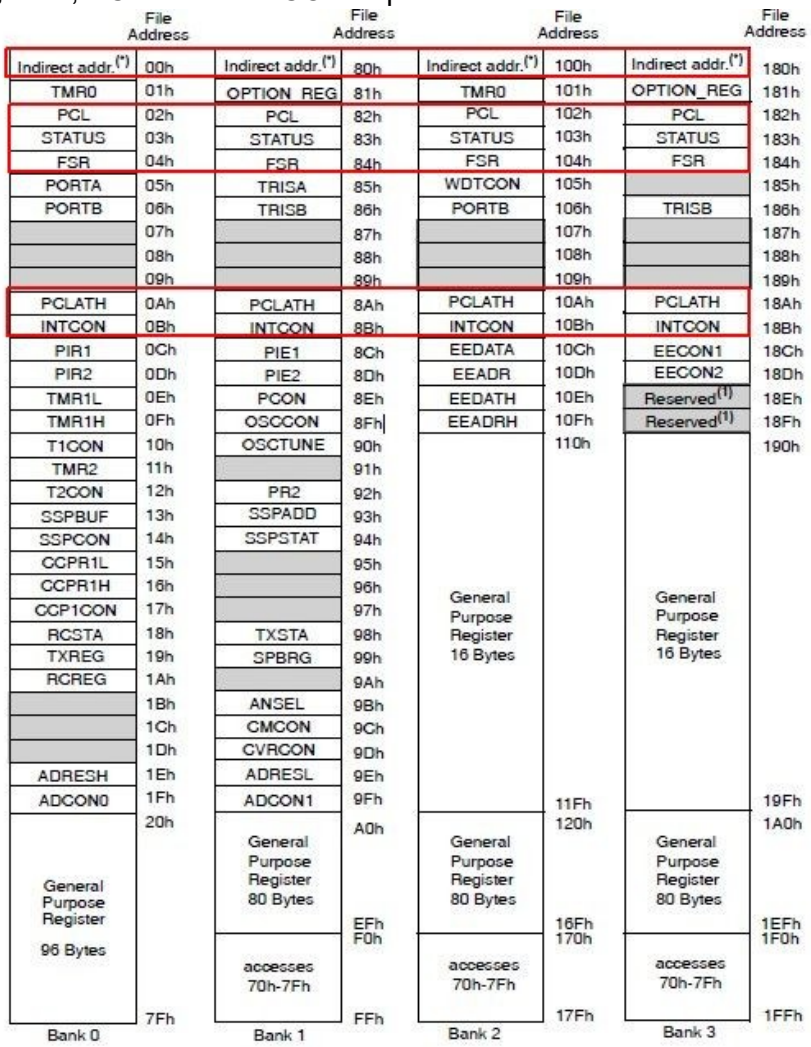
STATUS: ARITHMETIC STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

RP1 – bit 6 di STATUS	RP0 – bit 5 di STATUS	Banco di memoria accessibile
1	1	Bank3 (180 Hex – 1FF Hex)
1	0	Bank2 (100 Hex – 17F Hex)
0	1	Bank1 (80 Hex – FF Hex)
0	0	Bank0 (0 Hex – 7F Hex)

www.danielepostacchini.it

Ci sono altri registri che si trovano su tutti e 4 i banchi di memoria, questi sono accessibili sempre a prescindere dai bit RP0 ed RP1 e sono quelli evidenziati in figura. Pertanto si può leggere o scrivere sul registro Status, Indirect.addr., FSR, PCLATH e INTCON a prescindere dal banco di memoria selezionato.



La zona **General Purpose Register** (quella liberamente utilizzabile) non è molto grande, nel microcontrollore PIC 16F88 ad esempio, è complessivamente di **368 Byte**, ed è lo spazio che abbiamo per poter dichiarare delle variabili. Uno spazio molto piccolo se pensiamo agli 8 GB dei nostri PC, ma bisogna considerare che a differenza dei normali PC il microcontrollore viene utilizzato, nella gran parte dei casi, per eseguire un programma alla volta e per questo la memoria a disposizione è più che sufficiente. Ci sono comunque microcontrollori che hanno maggiori risorse ed un maggiore quantitativo di RAM.

In ogni microcontrollore è comunque presente una piccola EEPROM nel caso del PIC 16F88 essa ha una dimensione di 256 Byte.

Nella tabella seguente una sintesi delle caratteristiche del PIC e della dimensione della memoria interna.

Device	Program Memory		Data Memory		I/O Pins	10-bit A/D (ch)	CCP (PWM)	AUSART	Comparators	SSP	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)							
PIC16F87	7168	4096	368	256	16	N/A	1	Y	2	Y	2/1
PIC16F88	7168	4096	368	256	16	1	1	Y	2	Y	2/1

7168 Byte di memoria programma che può contenere 4096 istruzioni, 368 Byte di RAM e 256 Byte di EEPROM. Inoltre ci sono 16 pin di input/output, 1 convertitore A/D, un uscita PWM, una seriale USART, 2 comparatori, una porta seriale SSP, 2 timer da 8 bit ed uno da 16bit.

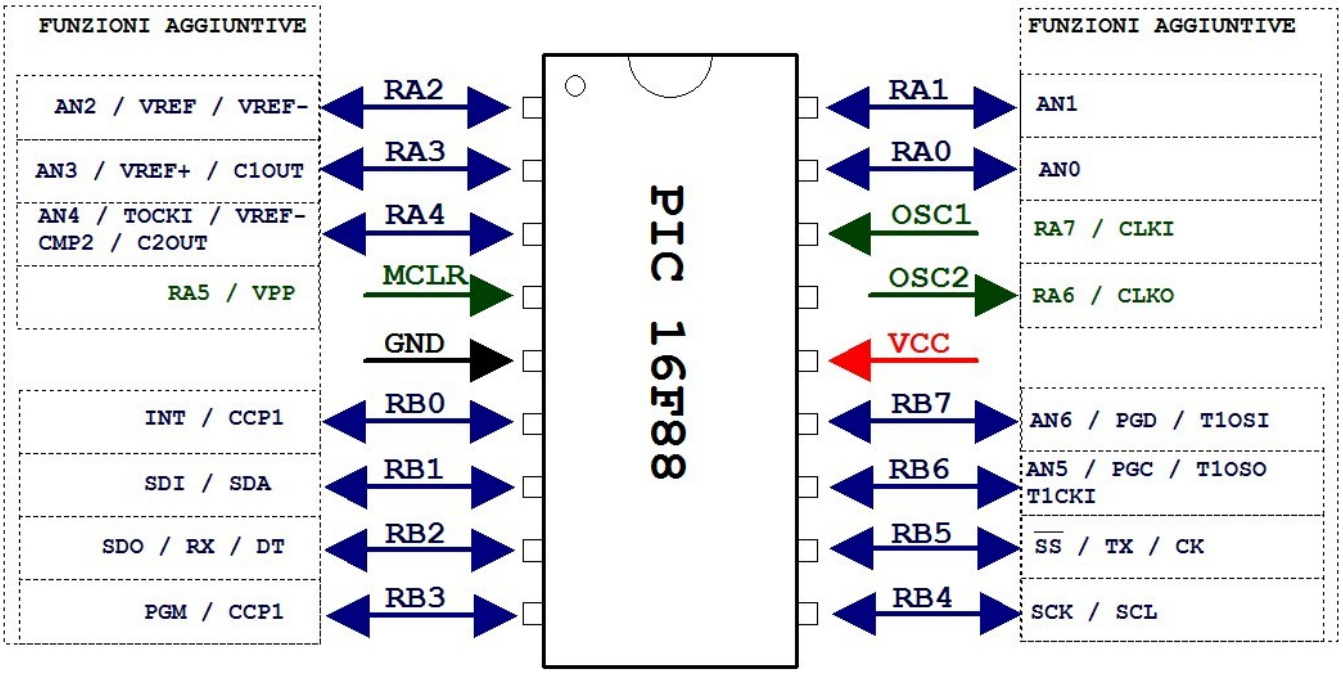
Approfondiremo il significato di PMW, USART e SSP in seguito.

www.danielepostacchini.it

GPIO (General Purpose Input Output)

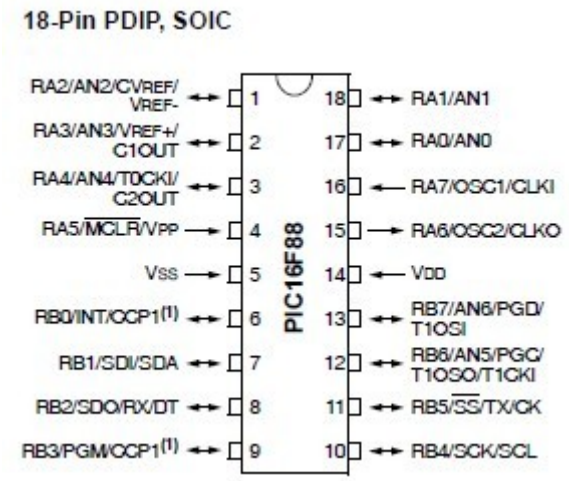
Fatta questa doverosa premessa, andiamo a fare un po' d'ordine su quanto detto, andando ad analizzare le parti essenziali per scrivere una prima semplice applicazione che potrebbe essere la gestione di un uscita o di un ingresso digitale.

Come abbiamo visto nell'immagine iniziale, ci sono anche dei pin per il collegamento di un clock esterno o per il segnale di reset (**MCLR**). Negli ultimi modelli di PIC, è presente un circuito interno per generare il clock, e per questo motivo i due pin **OSC1** e **OSC2** possono essere utilizzati come ingresso o uscita. Stessa cosa vale per il pin di reset, che se non utilizzato per questa funzione può diventare un bit di ingresso. **La direzione delle frecce sta ad indicare se il pin può essere bidirezionale, solo ingresso o solo uscita.**



Per definire se utilizzare così il clock interno, liberando così i piedini **OSC1** e **OSC2** o se non utlizzare il reset liberando il piedino **MCLR**, occorre agire sulla configurazione del PIC in fase di programmazione come vedremo più avanti.

In questa prima fase non considereremo le funzioni aggiuntive, ma ci lmiteremo ad utilizzare tutti i piedini come input o output digitali. **Per comodità utilizzeremo il clock interno e non utlizzeremo il reset MCLR, in modo da realizzare la prima esperienza con pochissimi componenti aggiuntivi.**



Piedinatura del PIC16F88 tratta dal data-sheet

www.danielepostacchini.it

Per gestire **PORTA** e **PORTB**, come piedini di ingresso/uscita, dobbiamo disattivare tutte le funzioni aggiuntive ad essi associate. Ad esempio su PORTA ci sono gli ingressi analogici che vanno poi nei due circuiti interni analogici che sono i **COMPARATORI** ed il **CONVERTITORE A/D**.

Per disattivare i COMPARATORI occorre agire sul registro **CMCON**, per disattivare il CONVERTITORE A/D invece occore agire sul registro **ANSEL.**

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h	WDTCON	185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h		87h		107h		187h	
08h		88h		108h		188h	
09h		89h		109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(1)
0Fh	TMR1H	8Fh	OSCCON	10Fh	EEADRH	18Fh	Reserved(1)
10h	T1CON	90h	OSCTUNE	110h	General Purpose Register 16 Bytes	190h	General Purpose Register 16 Bytes
11h	TMR2	91h					
12h	T2CON	92h	PR2				
13h	SSPBUF	93h	SSPADD				
14h	SSPCON	94h	SSPSTAT				
15h	CCPR1L	95h					
16h	CCPR1H	96h					
17h	CCP1CON	97h					
18h	RCSTA	98h	TXSTA				
19h	TXREG	99h	SPBRG				
1Ah	RCREG	9Ah					
1Bh		9Bh	ANSEL				
1Ch		9Ch	CMCON				
1Dh		9Dh	CVRCON				
1Eh	ADRESH	9Eh	ADRESL				
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
		EFh		16Fh		1EFh	
		F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h	accesses 70h-7Fh
7Fh		FFh		17Fh		1FFh	

Leggendo il data sheet nelle apposite sezioni, vediamo che per disattivare il convertitore A/D occorre mettere **tutti i bit del registro a 0**. Ogni bit del registro infatti **se è 1 attiva il reativo canale AN** di ingresso, se invece è 0 consente l'utilizzo dell'ingresso corrispondente come digitale.

ANSEL: ANALOG SELECT REGISTER (ADDRESS 9Bh) PIC16F88 DEVICES ONLY

U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0

Per disattivare i comparatori invece occorre mettere ad 1 i 3 bit CM2,CM1 e CM0, scrivendo nel registro **CMCON il valore 0x07**.

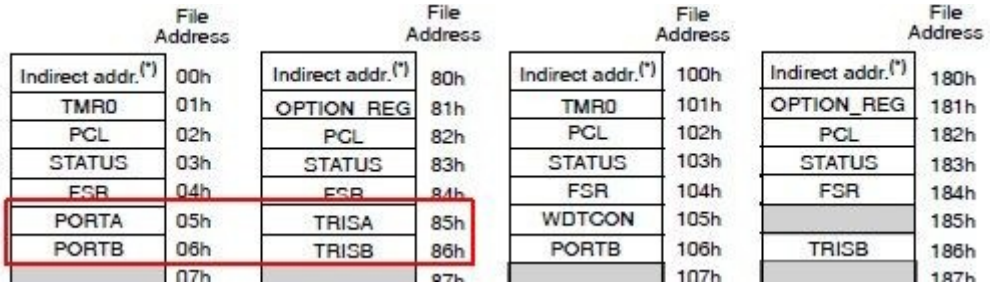
CMCON: COMPARATOR MODULE CONTROL REGISTER (ADDRESS 9Ch)

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

www.danielepostacchini.it

Una volta che PORTA e PORTB sono state configurate come ingressi/uscite digitali, occorre definire se sono ingressi o uscite. Ricordandosi sempre che quache piedino può essere solo ingresso (bit 5 e 7 di PORTA) o solo uscita (bit 6 di PORTA), gli altri pin potranno essere configurati in maniera indipendente senza alcuna restrizione.

Per fare questo occorre agire sui due registri TRISA e TRISB presenti nel banco di memoria 1.



Il funzionamento è molto semplice ad ogni bit del registro **TRIS** corrisponde un bit del corrisponente registro ***PORT***, se il bit del registro ***TRIS*** viene messo a 0 il corrispondente bit del registro **PORT** gestirà il piedino corrispondente come un uscita, in caso contrario come un ingresso.

BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TRISA	1	0	1	1	0	0	0	0
PORTA	IN	OUT	IN	IN	OUT	OUT	OUT	OUT

BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TRISB	1	1	1	1	0	0	0	0
PORTB	IN	IN	IN	IN	OUT	OUT	OUT	OUT

0=uscita 1=ingresso

Nel caso sopra ad esempio la PORTB ha i 4 bit meno significativi, 0,1,2,3 configurati come uscita perché i corrispondenti bit di TRISB stanno a 0.

I bit di PORTA e PORTB, vengono chiamati con la sigla RA o RB, e corrispondono fisicamente allo stato dei piedini del microcontrollore, perciò se mettessimo ad 1 il bit 0 di PORTB (RB0) troveremmo una tensione di 5 Volt sul piedino 0 di PORTB.

In pratica per portare un piedino ad un livello di tensione di 0 Volt o di 5 Volt, è sufficiente scrivere il valore binario 0 o 1 nel registro PORT corrispondente al piedino.

Stesso discorso per gli ingressi, volendo testare se un ingresso si trova a 0 Volt o a 5 Volt, è sufficiente leggere il valore binario nel registro PORT corrispondente al registro.

Tutto questo ovviamente dopo aver configurato correttamente i registri come finora descritto.

BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

www.danielepostacchini.it

Come abbiamo visto finora, i registri vengono identiicati con un nome che corrisponde ad un indirizzo di RAM, ad esempio **PORTB** corrisponde alla locazione 06 Hex della RAM. Ma anche i bit all'interno dei registri vengono identificati con un nome, es, **RP0** ed **RP1** del registro **STATUS** o **RA0** che è il bit 0 del registro **PORTA**.
L'associazione degli indirizzi dei registri o dei singoli bit con il nome corrispondente è fatta all'interno di un file, nel nostro caso P16f88.inc, che andrà incluso all'interno del nostro programma in modo da poter utilizzare i nomi anziché gli indirizzi.

Prima di procedere con la parte relativa alla programmazione, diamo qualche informazione in più su qualche altro registro.

REGISTRO STATUS

Questo registro contiene i seguenti flag:

STATUS: ARITHMETIC STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

- **IRP** serve per selezionare il banco di memoria se si utilizza l'indirizzamento indiretto
- **RP1** ed **RP0** servono per selezionare il banco di memoria attivo tra i 4 disponibili
- **TO** e **PD**, sono due bit legato alla gestione del watchdog
- **Z**,**DC** e **C** sono invece dei bit che segnalano i risultati delle operazioni eseguite dalla **A.L.U.** ad esempio **Z** è il flag di zero che va ad 1 se l'ultima istruzione eseguita da come risultato 0.

INDIRIZZAMENTO INDIRETTO Indirect Addr. e FSR

Questi due registi invece vengono utiizzati se si utiizza l'indirizzamento indiretto (Indirect Addressing).
Ci sono due modalità di accedere in lettura o in scrittura alla RAM, la prima è l'indirizzamento diretto che consiste nello scrivere o leggere direttamente il valore nel registro. La seconda modalità invece è l'indirizzamento indiretto, in questo caso si utilizzano i due registri sopra indicati; **IND (Indirect Addr.)** e **FSR**.
In pratica nel registro **FSR** si dovrà scrivere l'indirizzo fisico del registro che si vuol leggere o scrivere, mentre in **IND** va il suo contenuto.
Facciamo un esempio, se vogliamo scrivere in **PORTB** il valore 1F Hex, dovremmo scrivere prima in **FSR** il valore 06 Hex e successivamente in **IND** il valore 1F Hex.

Dovendo affrontare per ora i concetti basilari al fine di realizzare un semplice programma in assembly, approfondiremo in altro momento gli aspetti per ora non necessari.

www.danielepostacchini.it

PROGRAMMAZIONE DEI MICROCONTROLLORI PIC

Come in tutti i microprocessori, anche con il micrcontrollore si potrà scrivere un programma in un linguaggio evoluto come il C il Basic o altri, ma prima di passare a questo, proviamo a vedere qualche esempio con la programmazione in Assembly.

I microcontrollori hanno un set di istruzioni ridotto, per questo vengono definiti **R.I.S.C.** cioè **R**educted **S**et **I**nstruction **C**omputer. Il set di istruzioni di un PIC16F88 ad esempio conta 35 istruzioni di seguito elencate sinteticamente.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Possiamo notare che ogni istruzione necessita di 14bit, per questo motivo uno spazio di 7168 bytes della flash può contenere 4096 istruzioni.

Device	Program Flash	Data Memory	Data EEPROM
PIC16F87/88	4K x 14	368 x 8	256 x 8

Device	Program Memory		Data Memory	
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)
PIC16F87	7168	4096	368	256
PIC16F88	7168	4096	368	256

Ad ogni istruzione in assembly corrisponde il codice binario che rappresenta il codice macchina e che sarà scritto nella Flash del microcontrollore tramite il dispositivo programmatore.

www.danielepostacchini.it

Nella precedente tabella, si riporta inoltre la durata in termini temporali di ogni istruzione, indicando il numero di "cicli istruzione" nella colonna "cycles" necessari per ogni istruzione.

Un ciclo istruzione è pari a 4 cicli di clock, pertanto se la frequenza del clock fosse di 4MHz avremmo che:

$$ciclo\,istruzione = 4 * \frac{1}{clock\,frequency} = 4 * \frac{1}{4\,MHz} = 1\,usec$$

Un'istruzione BCF perciò durerà 1 ciclo istruzione e cioè 1usec, o un'istruzione GOTO durerà 2 cicli di clock e cioè 2usec.

Realizzando dei cicli ripetitivi, si potranno pertanto realizzare delle routine con ritardi noti.

Bisogna però fare attenzione ad un aspetto, in diversi PIC, come ad esempio nel PIC16F88, ci sono dei registri interni che permettono di agire sulla frequenza di clock, andando ad operare su dei circuiti interni, come MUX e PLL.

Nel caso del PIC16F88 ad esempio, ci sono due registri che agiscono sul clock e sono: **OSCTUNE** e **OSCCON.**

Nel capitolo 4.0 del datasheet, troviamo la spiegazione di questi due registri.

OSCTUNE

REGISTER 4-1: OSCTUNE: OSCILLATOR TUNING REGISTER (ADDRESS 90h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	TUN5	TUN4	TUN3	TUN2	TUN1	TUN0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-0 **TUN<5:0>:** Frequency Tuning bits

011111 = Maximum frequency
011110 =
•
•
•
000001 =
000000 = Center frequency. Oscillator module is running at the calibrated frequency.
111111 =
•
•
•
100000 = Minimum frequency

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Questo registro permette di modificare la frequenza di clock impostata fino una percentuale massima del 12,5%. Il registro si utilizza per calibrare la frequenza del clock quando necessario, come ad esempio quando è selezionato una sorgente di clock interna senza quarzo esterno.

Il valore di default è pari a 0.

www.danielepostacchini.it

OSCCON

REGISTER 4-2: OSCCON: OSCILLATOR CONTROL REGISTER (ADDRESS 8Fh)

U-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS(1)	IOFS	SCS1	SCS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-4 **IRCF<2:0>:** Internal RC Oscillator Frequency Select bits
000 = 31.25 kHz
001 = 125 kHz
010 = 250 kHz
011 = 500 kHz
100 = 1 MHz
101 = 2 MHz
110 = 4 MHz
111 = 8 MHz

bit 3 **OSTS:** Oscillator Start-up Time-out Status bit(1)
1 = Device is running from the primary system clock
0 = Device is running from T1OSC or INTRC as a secondary system clock

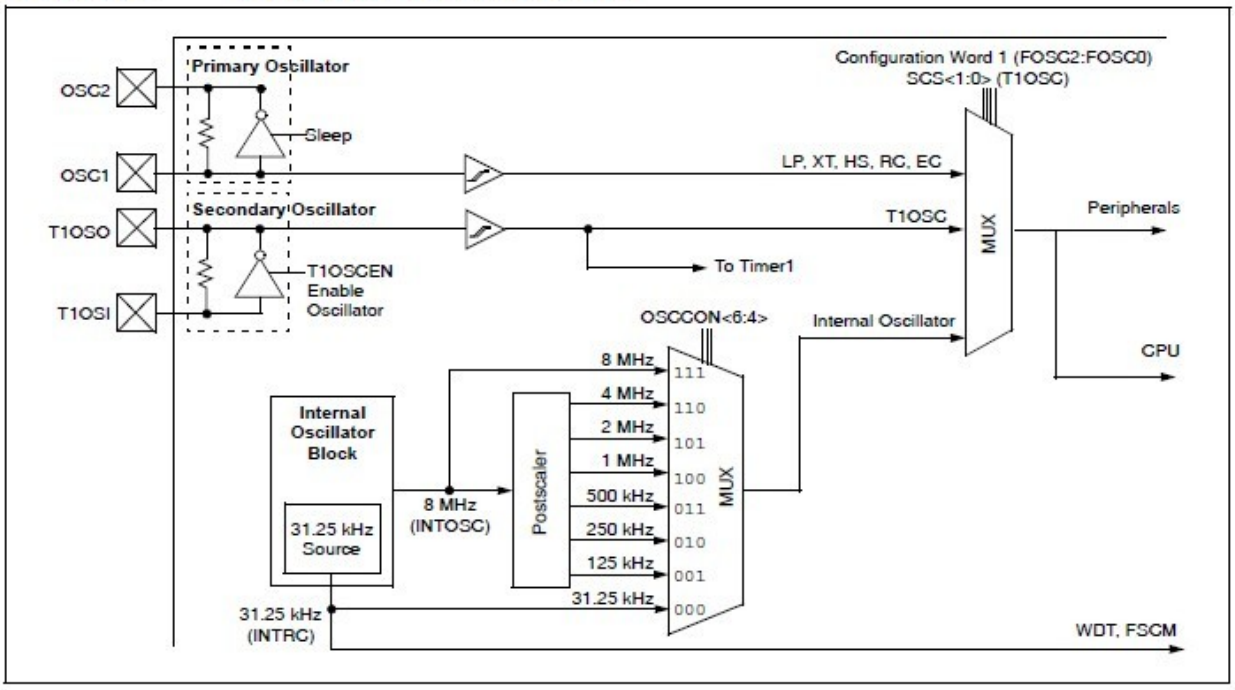
Note 1: Bit resets to '0' with Two-Speed Start-up mode and LP, XT or HS selected as the oscillator mode.

bit 2 **IOFS:** INTOSC Frequency Stable bit
1 = Frequency is stable
0 = Frequency is not stable

bit 1-0 **SCS<1:0>:** Oscillator Mode Select bits
00 = Oscillator mode defined by FOSC<2:0>
01 = T1OSC is used for system clock
10 = Internal RC is used for system clock
11 = Reserved

Il valore di default di questo registro è pari a 0, ciò significa che i 3 bit denominati IRCF sono tutti a 0.
I 3 bit lavorano su un multiplexer interno, ed il valore 000, sta ad indicare che la frequenza di clock non sarà 8MHz ma 31,25kHz. Per ottenere 8MHz i 3 bit devono stare ad 1.

FIGURE 4-6: PIC16F87/88 CLOCK DIAGRAM

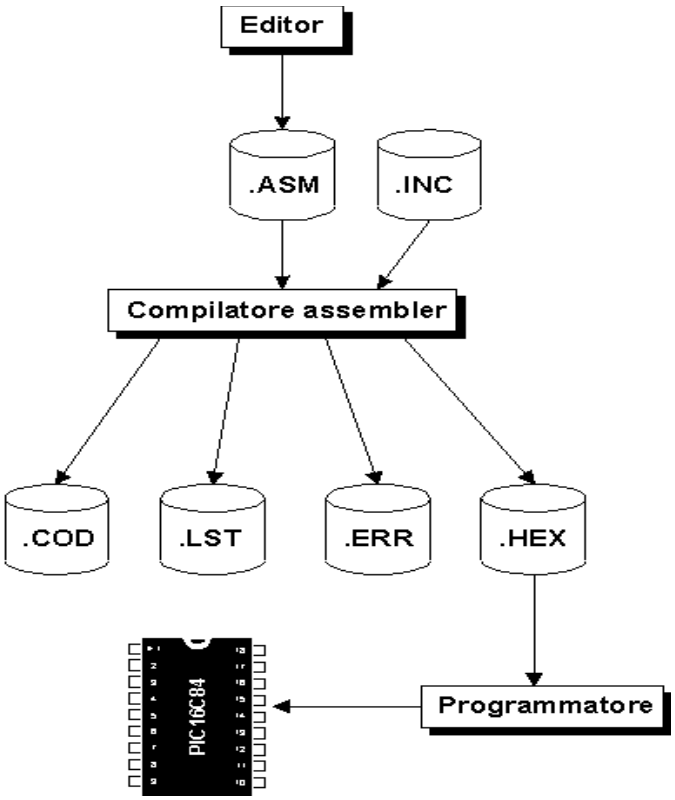


Il significato degli altri bit del registro è facilmente intuibile e senza scendere nel dettaglio, possiamo affermare che per mantenere la frequenza di clock dell'oscillatore (interno o esterno) bisogna impostare OSCON=0xFC (valore in esadecimale).

www.danielepostacchini.it

PROGRAMMAZIONE ASSEMBLY DEL MICROCONTROLLORE PIC

Il programma una volta scritto nel linguaggio assembler, tramite un compilatore assembler (assemblatore) viene convertito in un file binario (.HEX) contenente il codice macchina, che può essere caricato nella memoria programma del microcontrollore in modo da essere eseguito al primo avvio.



In file **ASM** è quello scritto da chi fa il programma in linguaggio assembler. Il file **INC** è quello descritto prima, che contiene le associazioni tra nome dei registri e riga di memoria (P16F88.inc).
Il file **HEX** contiene il file binario da caricare nel microcontrollore.
Gli altri sono file non indispensabili, ma utili, ad esempio il file .**ERR** contiene tutti gli errori di compilazione, o il file .**LST** contiene il listato del programma con il codice macchina e le righe di memoria corrispondenti.

www.danielepostacchini.it

ISTRUZIONI E DIRETTIVE

Innanzitutto bisogna fare una distinzione tra istruzioni e direttive. Le prime sono indirizzate al microcontrollore, e come abbiamo già detto vengono scritte con un codice binario nella memoria Flash.
Le seconde sono invece indirizzate al compilatore assembler, non sono istruzioni che il microprocessore eseguirà. Per essere più chiari analizziamo le 3 parti del seguente programma di esempio.
Le prime istruzioni che vengono date sono per il compilatore assembler e sono le **DIRETTIVE**.
Di seguito vediamo la direttiva **INCLUDE** che aggiunge al nostro programma il file **P16f88.inc**.
Senza di questa direttiva scrivendo ad esempio PORTB nel programma riceveremmo degli errori in quanto dovremo scrivere l'indirizzo di PORTB e non il nome. Nel file P16F88.inc ci sono tutte le assegnazioni NOME-INDIRIZZO, o NOME-BIT, in modo che durante la programmazione si potranno utilizzare i nomi dei registri e dei loro bit al posto del loro indirizzo fsico.

Un altra direttiva è EQU che assegna alla label LED il valore 0, in pratica con questa direttiva si dice al compilatore di sostituire alla parla LED il numero 0.

Anche CBLOCK può essere intesa come una direttiva per il compilatore, in quanto ci permette di definire delle variabili da 8 bit. Nel programma dopo la direttiva CBLOCK 0x20 vengono dichiarate delle variabili con la seguente sintassi:

Count: 2

In pratica si dice che l'indirizzo di RAM 0x20 corrrisponde a Count e l'indirizzo 0x21 corrisponde a Count+1.
I numeri esadecimali possono essere rappresentati con la dicitura **0x** come nel linguaggio C.

```
1   ;*************************************************************************
2           ;DIRETTIVE PER IL COMPILATORE/ASSEMBLATORE
3
4           #include    "P16F88.INC"    ;definizione del tipo di microcontrollore
5                                       ;viene incluso un file che contiene le
6                                       ;associazioni tra indirizzo fisico e nome
7                                       ;delle porte e dei bit del microcontrollore
8
9           ;DEFINIZIONE DI COSTANTI
10  LED     EQU         0               ;si assegna alla label LED la corrispondenza
11                                      ;al valore 0. LED non è un variabile.
12
13          ;DEFINIZIONE DELLE VARIABILI
14          CBLOCK   0x20               ;si identifica l'inizio della RAM libera
15          Count:2                     ;si dichiara una variabile Count e si riserva
16                                      ;per questa variabile due byte di RAM
17                                      ;da 8 bit, avremo che
18                                      ;indirizzo 0x20 = Count
19                                      ;indirizzo 0x21 = Count+1
20          ENDC                        ;fine della definizione di variabili
21
```

www.danielepostacchini.it

Dopo aver dato le direttive al compilatore possiamo scrivere il programma vero e proprio. Dobbiamo ricordarci che il microcontrollore all'accensione esegue l'istruzione contenuta nell'indirizzo di memoria Flash 0x00.
Il comando ORG è anch'esso una direttiva per il compilatore e ci indica dove scrivere le successive istruzioni.
Come si può vedere la prima istruzione è un salto, ad una label che si chiama "Inizio", e sempre tramite la direttiva ORG 0x100 associamo la label Inizio all'indirizzo di memoria Flash 0x100. Potevamo comunque scrivere il programma di seguito al primo indirizzo 0x00, ma conviene sempre saltare l'indirizzo 0x04 che viene utilizzato dall'interrupt del microcontrollore.

```
22      ;*************************************************************************
23              ;PROGRAMMA CONTENUTO NELLA FLASH
24              ORG         0x00            ;inizio del programma, il microcontrollore
25                                          ;inizierà a decodificare le istruzioni
26                                          ;partendo dalla locazione 0x00 della FLASH
27              GOTO        Inizio          ;prima istruzione, salto alla locazione
28                                          ;indicata dalla label "Inizio"
29              ORG         0x100           ;locazione di memoria dove è scritto il
30                                          ;programma
31      Inizio                              ;label inizio, posizionata alla riga 0x100
32              ;scelta del banco di memoria dove scrivere, BANK1
33              bsf         STATUS,RP0      ;metto ad 1 il bit RPO del registro STATUS
34              bcf         STATUS,RP1      ;metto ad 0 il bit RP1 del registro STATUS
35              ;definisco tutti i pin di PORTA come digitali
36              movlw       0x00            ;carico il valore 0x00 in W
37              movwf       ANSEL           ;copio il contenuto di W nel registro ANSEL
38              ;dIsabilito i comparatori
39              movlw       0x07            ;carico il valore 0x07 in W
40              movwf       CMCON           ;copio il contenuto di W nel registro CMCON
41              ;seleziono il clock primario ad 8MHZ
42              movlw       0xFC            ;carico il valore 0xFC in W
43              movwf       OSCCON          ;copio il contenuto di W nel registro OSCCON
44              ;definisco tutti i bit di PORTA come ingressi tranne RA6
45              movlw       0xBF            ;carico il valore 0xBF in W
46              movwf       TRISA           ;copio il contenuto di W nel registro TRISA
47              ;definisco tutti i bit di PORTB come uscita
48              movlw       0x00            ;;carico il valore 0x00 in W
49              movwf       TRISB           ;copio il contenuto di W nel registro TRISB
50              ;scelta del banco di memoria dove scrivere, BANK0
51              bcf         STATUS,RP0
52              bcf         STATUS,RP1
53              ;inizializzazione dello stato del led
54              bsf         PORTB,LED       ;metto ad 1 il bit 0 di PORTB
55
56              ;loop del programma
57      MainLoop                            ;label che identifica il loop del programma
58              call        Delay           ;chiamata alla subroutine Delay
59              btfsc       PORTB,LED       ;test bit0 di PORTB se 0 salto prossima riga
60              goto        SetToZero       ;salto alla label SetToZero
61              bsf         PORTB,LED       ;metto ad uno il bit o di PORTB
62              goto        MainLoop        ;salto alla label MainLoop
63      SetToZero                           ;label SetToZero
64              bcf         PORTB,LED       ;resetto il bit 0 di PORTB
65              goto        MainLoop        ;salto alla labe MainLoop
66
```

www.danielepostacchini.it

```
67  ;*****************************************************************************
68          ;SUBROUTINE Delay
69
70  Delay                               ;label inizio subroutine
71          clrf        Count           ;azzero il contenuto della variabile Count
72          clrf        Count+1         ;azzero il contenuto della variabile Count+1
73  DelayLoop                           ;label DelayLoop
74          decfsz      Count,1         ;decremento la variabile Count
75          goto        DelayLoop       ;salto alla label DelayLoop
76          decfsz      Count+1,1       ;decremento la variabile Count+1
77                                      ;se zero salto la prossima istruzione
78          goto        DelayLoop       ;salto alla label DelayLoop
79          return                      ;ritorno alla riga di programma dove
80                                      ;è stata chiamata la subroutine
81  ;*****************************************************************************
82
83          END                         ;fine codice
```

Il programma sopra scritto, farà lampeggiare un led collegato al pin RB0. Possiamo comprendere il funzionamento del programma leggendo i commenti.

Per scrivere un programma in assembly, occorre rispettare delle regole sulla scrittura. Il programma dovrà essere divso in 4 colonne, la prima per le LABEL, la seconda per le ISTRUZIONI, la terza per gli OPERANDI, e l'ultima per i COMMENTI, che in un programma assembly vanno preceduti dal punto e virgola.

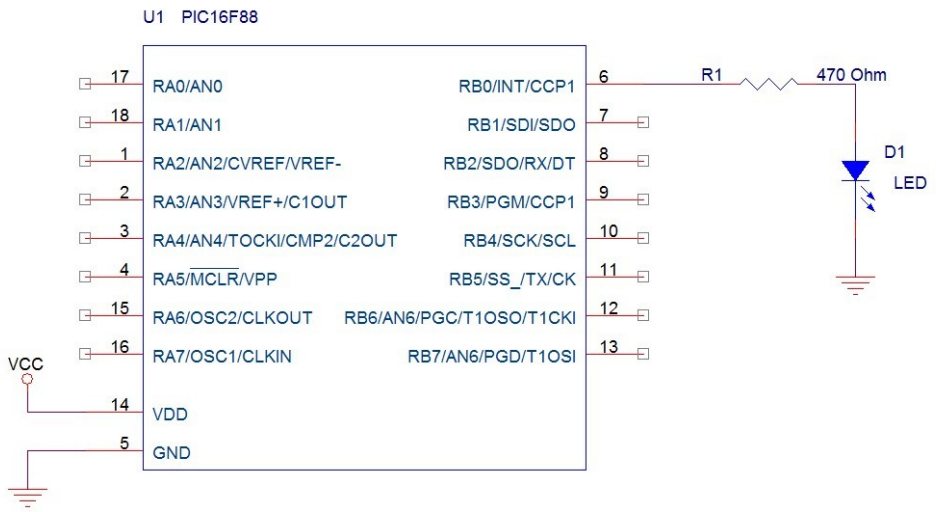
LABEL	ISTRUZIONE	OPERANDI	COMMENTI

Nel programma troviamo una prima parte che serve per definire i bit di PORTA e PORTB, nel nostro caso come input/output.

Poi c'è il loop del programma cioè il ciclo continuo all'interno del quale viene effettuato il lampeggio del LED.

Al termine c'è la subroutine, che in questo caso è una funzione di ritardo (DELAY) che serve per mantenere il LED acceso o spento per un tempo dipendente dal valore di Count e Count+1.

Lo schema di collegamento da utilizzare con questo programma è il seguente, Vdd=5Vdc.



www.danielepostacchini.it

Approfondiamo ora le istruzioni utilizzate.

ISTRUZIONE	ESEMPIO	SIGNIFICATO
MOVLW (Move Literal to W)	MOVLW 05H	Carica il valore 05 (esadecimale) nel registro accumulatore W
MOVWF (Move W to File register)	MOVWF PORTB	Carica il contenuto dell'accumulatore W nel registro indicato. Nell'esempio W va in PORTB
BCF (Bit Clear File register)	BCF PORTB,2	Mette a zero un bit di un registro. Nell'esempio mette a zero il bit 2 del registro PORTB
BSF (Bit Set File register)	BSF PORTB,2	Mette a uno un bit di un registro. Nell'esempio mette a uno il bit 2 del registro PORTB
BTFSC (Bit Test File register Skip if Clear)	BTFSC PORTB,2	Testa un bit di un registro, se è zero salta l'istruzione successiva. Nell'esempio testa il bit 2 di PORTB.
BTFSS (Bit Test File register Skip if Set)	BTFSS PORTB,2	Testa un bit di un registro, se è uno salta l'istruzione successiva. Nell'esempio testa il bit 2 di PORTB.
DECF (DECrement File register)	DECF PORTB,1 DECF PORTB,0	Decrementa di uno il valore contenuto nel registo indicato, e mette il risultato nello stesso registo o nell'accumulatore W. Nel primo esempio il valore contenuto in PORTB viene decrementato di uno ed il risultato rimane in PORTB. Nel secondo esempio il risultato va in W.
INCF (INCrement File register)	INCF PORTB,1 INCF PORTB,0	Incrementa di uno il valore contenuto nel registo indicato, e mette il risultato nello stesso registo o nell'accumulatore W. Nel primo esempio il valore contenuto in PORTB viene incrementato di uno ed il risultato rimane in PORTB. Nel secondo esempio il risultato va in W.
DECFSZ (DECremente File register Skip if Clear)	DECFSZ PORTB,1 DECFSZ PORTB,0	Decrementa il valore contenuto nel registro indicato, come nel caso precedente. Oltre a questo viene testato il risultato, se vale 0 salta l'istruzione successiva.
INCFSZ (INCrement File register Skip if Clear)	INCFSZ PORTB,1 INCFSZ PORTB,0	Incrementa il valore contenuto nel registro indicato, come nel caso precedente. Oltre a questo viene testato il risultato, se vale 0 salta l'istruzione successiva.
NOP (No OPeration)	NOP	Nessuna operazione
CALL	CALL DELAY	Chiama una subroutine o sottoprogramma. Nell'esempio chiama un sottoprogramma chiamato DELAY.
RETURN	RETURN	Ritorno dal sottoprogramma nel punto in cui è stato chiamato
CLRF (CleaR File register)	CLRF PORTB	Azzera il contenuto di un registro, nell'esempio viene messo a zero il contenuto di PORTB.
CLRW (CleaR W)	CLRW	Azzera il contenuto di W.
RRF (Rotate Right File register)	RRF PORTB,1 RRF PORTB,0	Ruota il registro indicato di una posizione a destra. Il risultato può essere messo nello stesso registro o in W. Nel primo esempio il risultato rimane in PORTB, nel secondo caso va in W.
RLF (Rotate Left File register)	RLF PORTB,1 RLF PORTB,0	Ruota il registro indicato di una posizione a sinistra. Il risultato può essere messo nello stesso registro o in W. Nel primo esempio il risultato rimane in PORTB, nel secondo caso va in W.

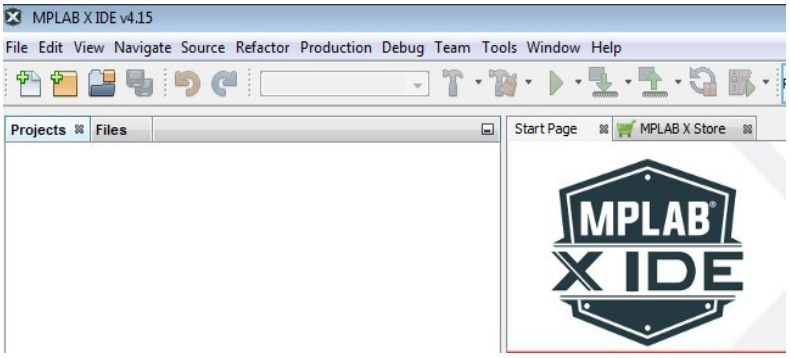
www.danielepostacchini.it

AMBIENTE DI SVILUPPO MPLAB

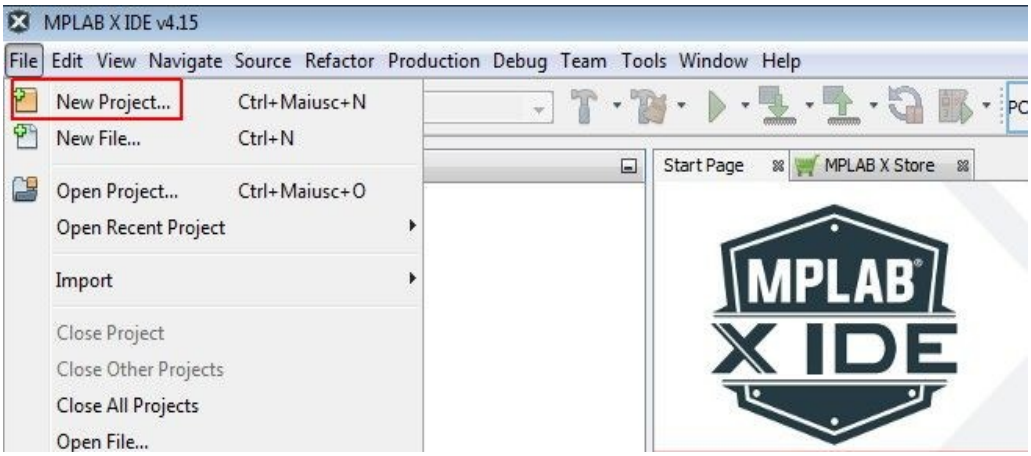
SCRITTURA E COMPILAZIONE DI UN PROGRAMMA

Per scrivere un programma in assembler, c'è la suite fornita dalla Microchip MPLAB. Il programma può anche essere scritto in linguaggi più evoluti come il linguaggio C, ma analizziamo i passi necessari per impostare e simulare un programma scritto in assembler.
La suite MPLAB è liberamente scaricabile dal sito della microchip, ma per utilizzare la programmazione in assembler si consiglia di scaricare la versione 5.35 al seguente link:
https://drive.google.com/file/d/1Jyo87jSOcKXP-1cNFFlOn7y5IJiSJevy/view?usp=sharing

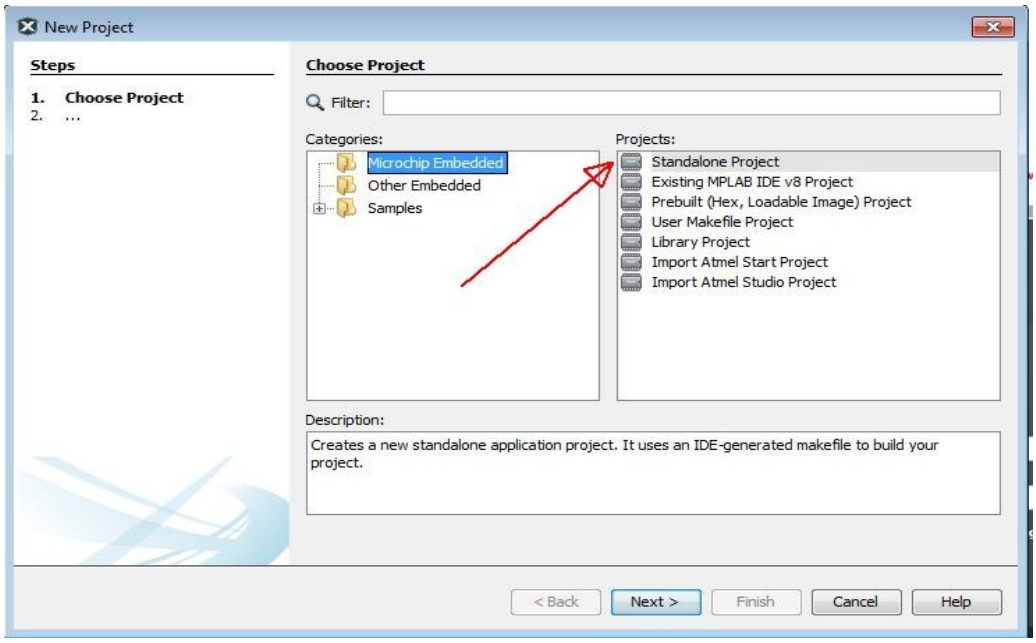
all'avvio del programma troviamo questa schermata:



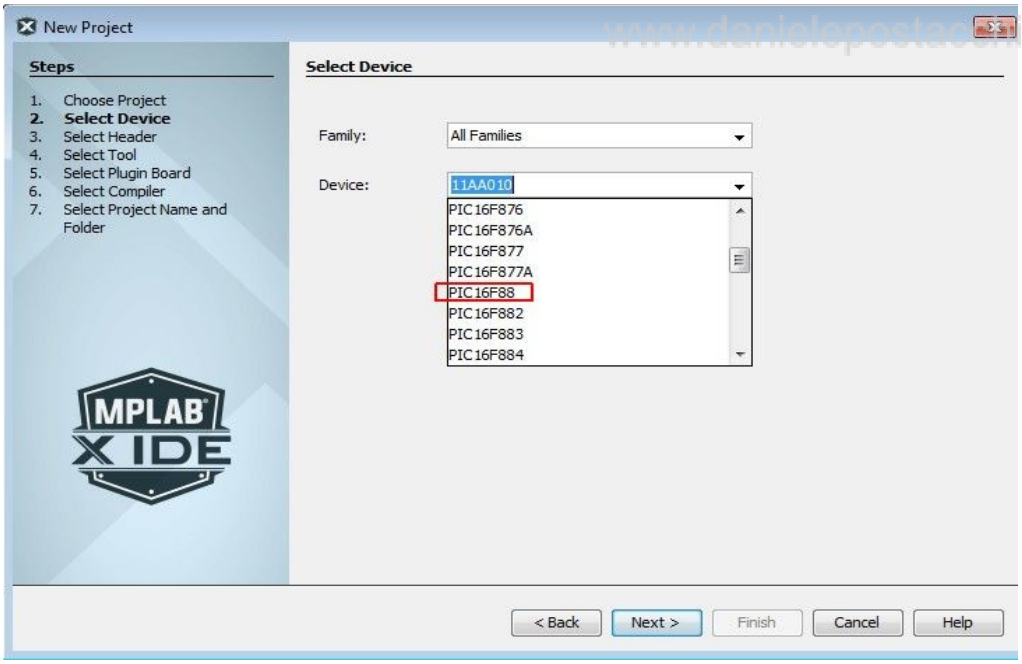
Proviamo a creare un nuovo progetto, ***File-New Project***



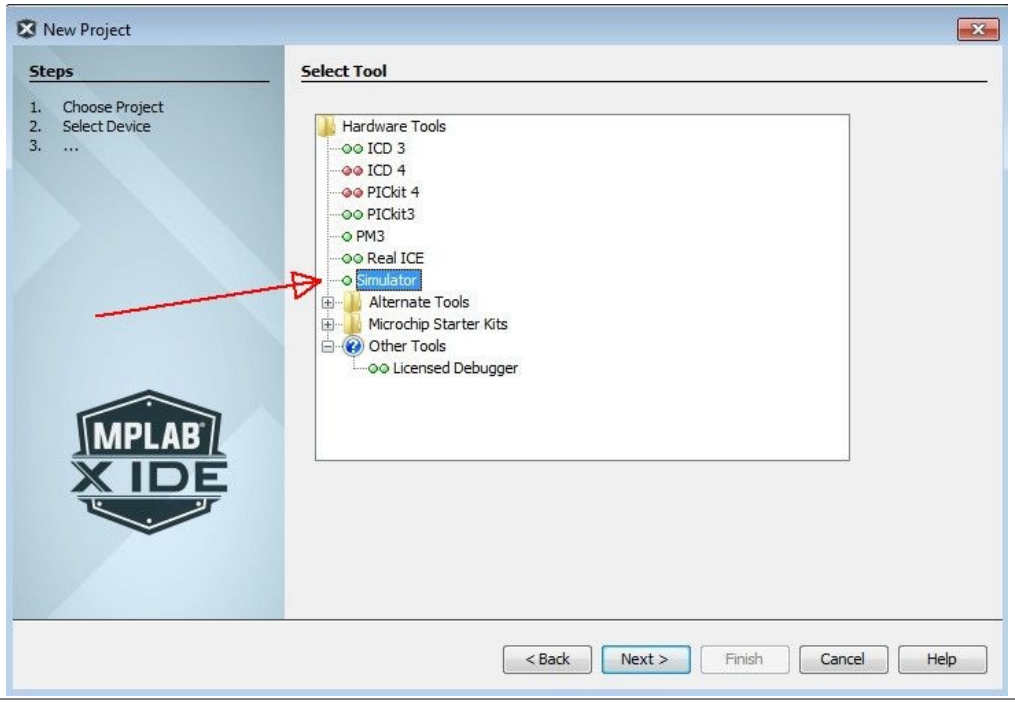
Scegliamo l'opzione
STANDALONE PROJECT



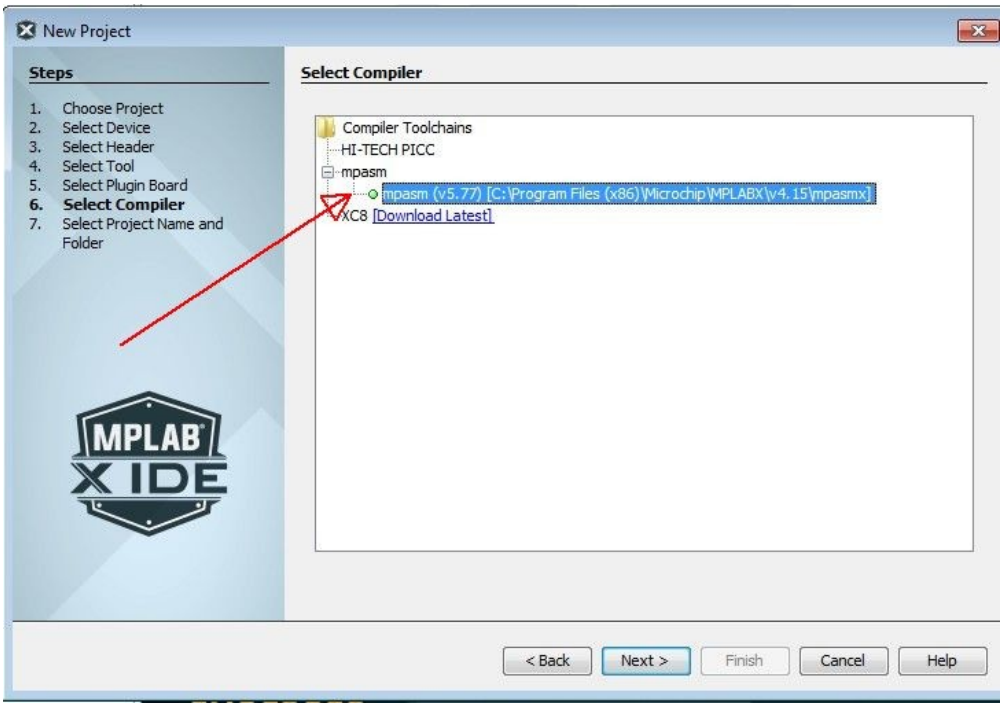
Scegliamo il tipo di microcontrollore, nel nostro caso il **PIC16F88**.



Scegliamo come strumento hardware il **simulatore**

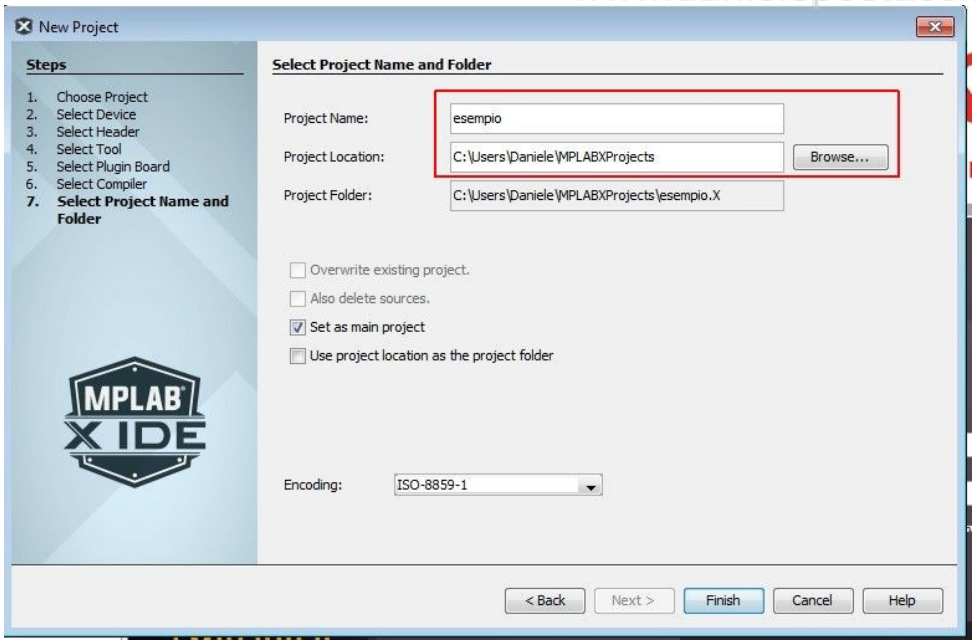


Selezioniamo il compilatore **MPASM**

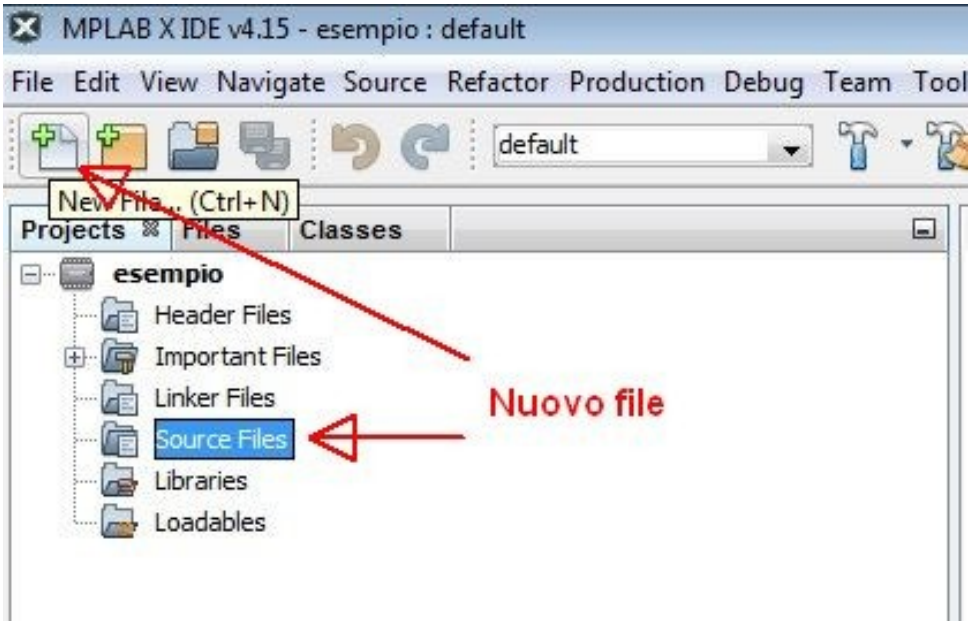


www.danielepostacchini.it

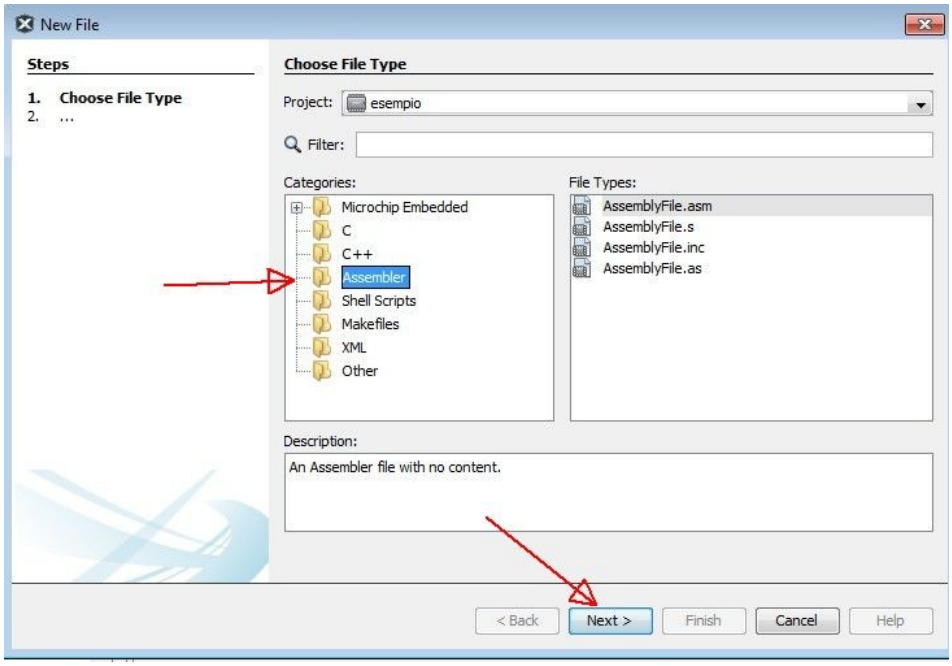
A questo punto occorre scegliere la cartella dove risiede il progetto ed il nome del progetto



A questo punto nella parte destra troveremo la struttura del progetto e come prima cosa dovremo aggiungere un nuovo file sorgente

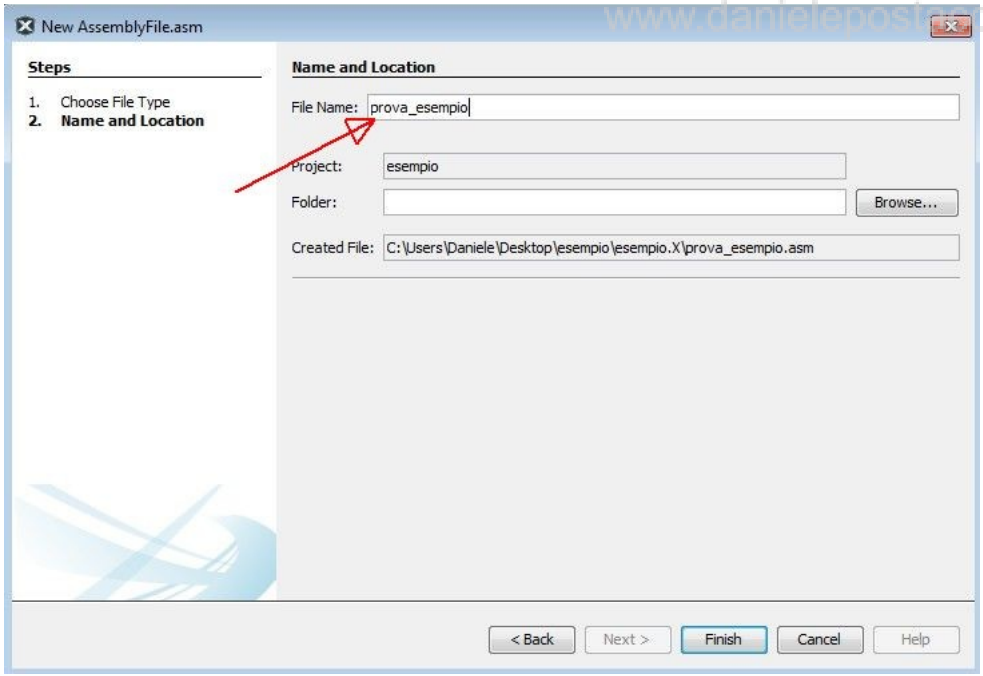


Bisogna selezionare linguaggio Assembler

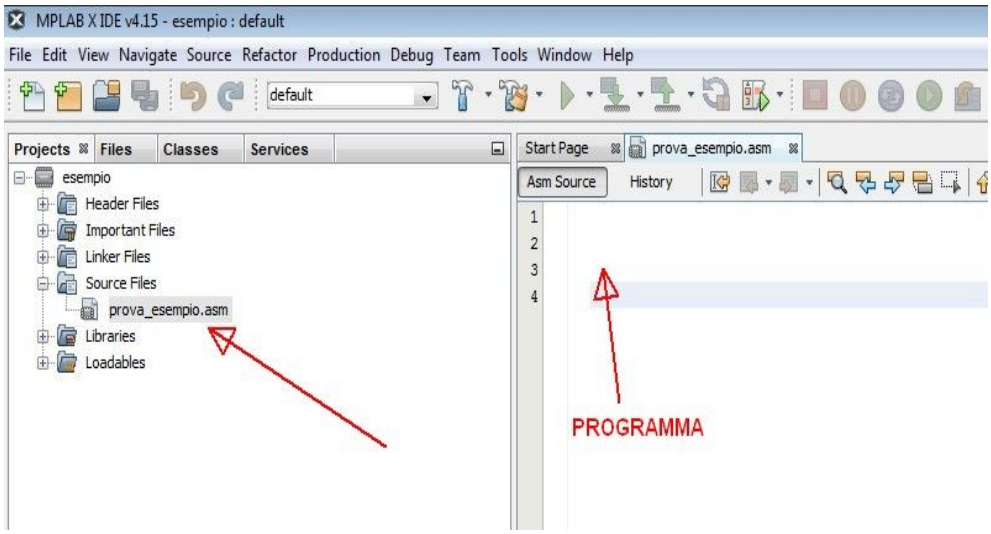


www.danielepostacchini.it

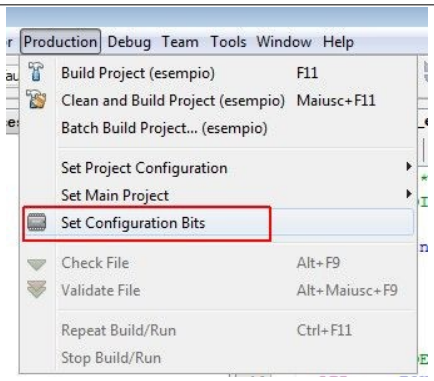
Successivamente occorre dare un nome al file .asm



Al termine avremo il nostro file .asm nella finestra del progetto a sinistra, mentre a destra potremo scrivere il programma.



Nel menù a tendina troviamo la configurazione del PIC, facciamo click con il mouse ed impostiamo sulla finestra in basso la configurazione come nella figura in basso.



Address	Name	Value	Field	Option	Category
2007	CONFIG1	FFD0	FOSC	INTOSCIO	Oscillator Selection bits
			WDTE	OFF	Watchdog Timer Enable bit
			PWRTE	ON	Power-up Timer Enable bit
			MCLRE	OFF	RA5/MCLR/VPP Pin Function Select bit
			BOREN	ON	Brown-out Reset Enable bit
			LVP	ON	Low-Voltage Programming Enable bit
			CPD	OFF	Data EE Memory Code Protection bit
			WRT	OFF	Flash Program Memory Write Enable bits
			CCPMX	RB0	CCP1 Pin Selection bit
			CP	OFF	Flash Program Memory Code Protection bit
2008	CONFIG2	FFFF	FCMEN	ON	Fail-Safe Clock Monitor Enable bit
			IESO	ON	Internal External Switchover bit

www.danielepostacchini.it

Analizziamo il significato dei vari campi della configurazione:

FOSC Possiamo scegliere varie opzioni sul tipo di clock, nel nostro caso abbiamo scelto con INOTSCIO, un clock interno in modo da lasciare i due piedini disponibili per in e out.

WDTE Watchdog, nel nostro caso lo teniamo disabilitato, altrimenti va gestito nel software.

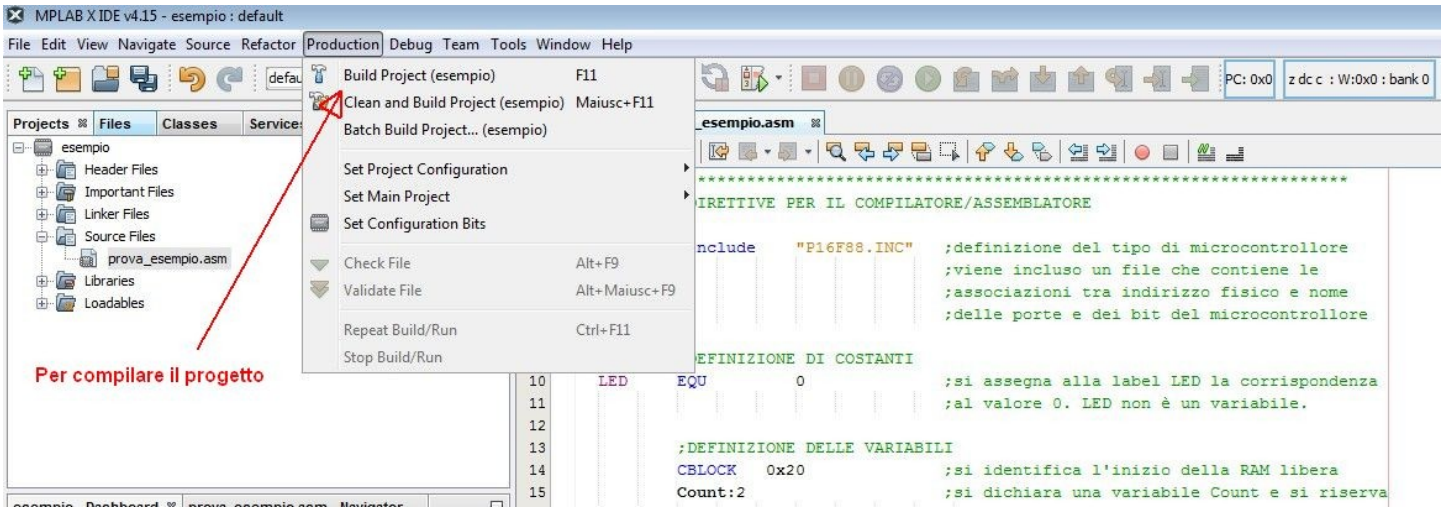
PWRTE E' un ritardo all'accensione del microcontrollore, lo attiviamo in modo che il micro avvierà l'esecuzione del programma quando la tensione di alimentazione sarà stabile.

MCLRE Lo teniamo disattivato in modo da utilizzare il pin di reset come ingresso.

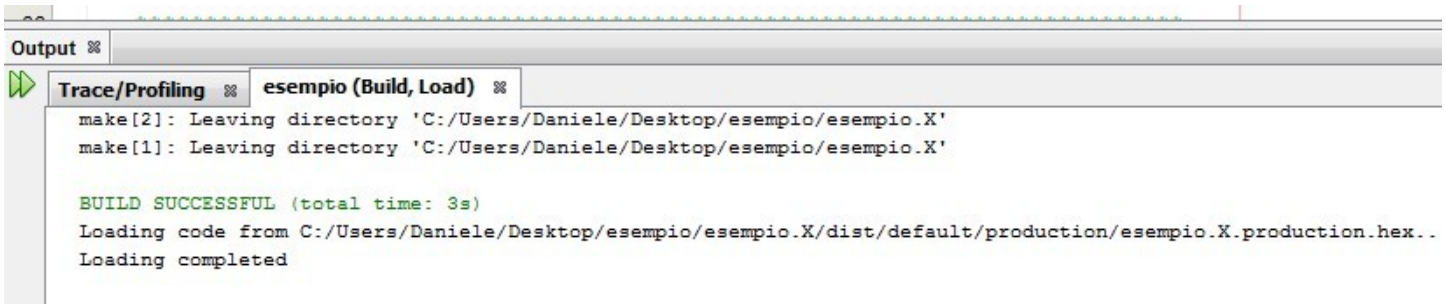
BOREN E' un controllo sull'alimentazione, se questa scende sotto un certo valore il micro viene resettato, possiamo attivarlo.

Per ora non consideriamo gli altri campi.

Al termine della scrittura del file possiamo compilare il tutto dal menù a tendina indicato, con BUILD PROJECT



Se non ci sono errori al termine nell finestra in basso troviamo il seguente messaggio BUILD SUCCESSFUL



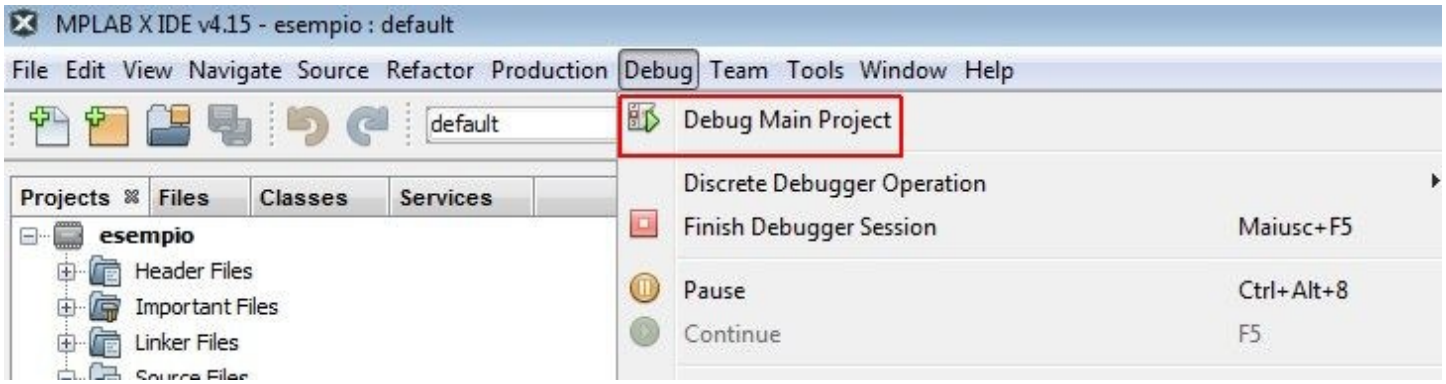
Al termine della compilazione verranno creati diversi file, vediamo i più importanti.
Entriamo nella cartella del progetto e proseguiamo con il percorso ***build-default-production***, qui troviamo il file con estensione **.err** che contiene gli errori, i warning ed i messaggi ottenuti in fase di compilazione.
Insieme ad esso troviamo i file con estensione **.lst** che invece contiene il codice assembly con i corrispondente codice esadecimale (codice macchina) e l'indirizzo di memoria dove esso risulta allocato.

Sempre nella cartella del progetto ma proseguendo con il percorso **dist-*default-production***, possiamo trovare il file più importante e cioè quello con estensione **.hex. Questo file dovrà essere caricato tramite un programmatore nella memoria del microcontrollore.**

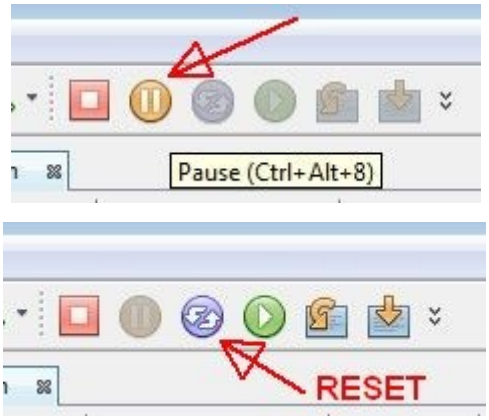
www.danielepostacchini.it

SIMULAZIONE DEL PROGRAMMA

Per avviare a simulazione fare clic su **Debug Main Project**



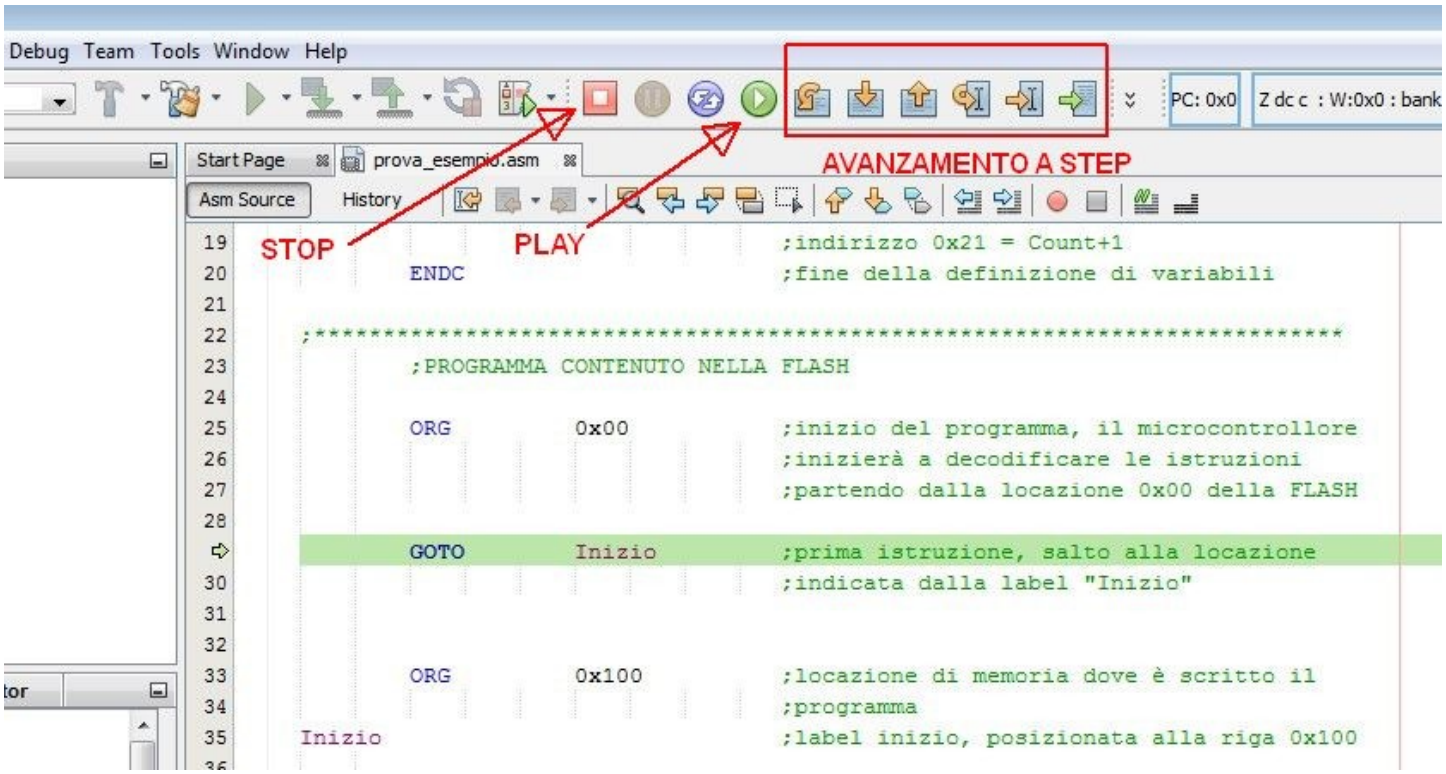
La simulazione è ora avviata e la prima cosa da fare è quella di metterla in pausa e successivamente resettare il programma.



---

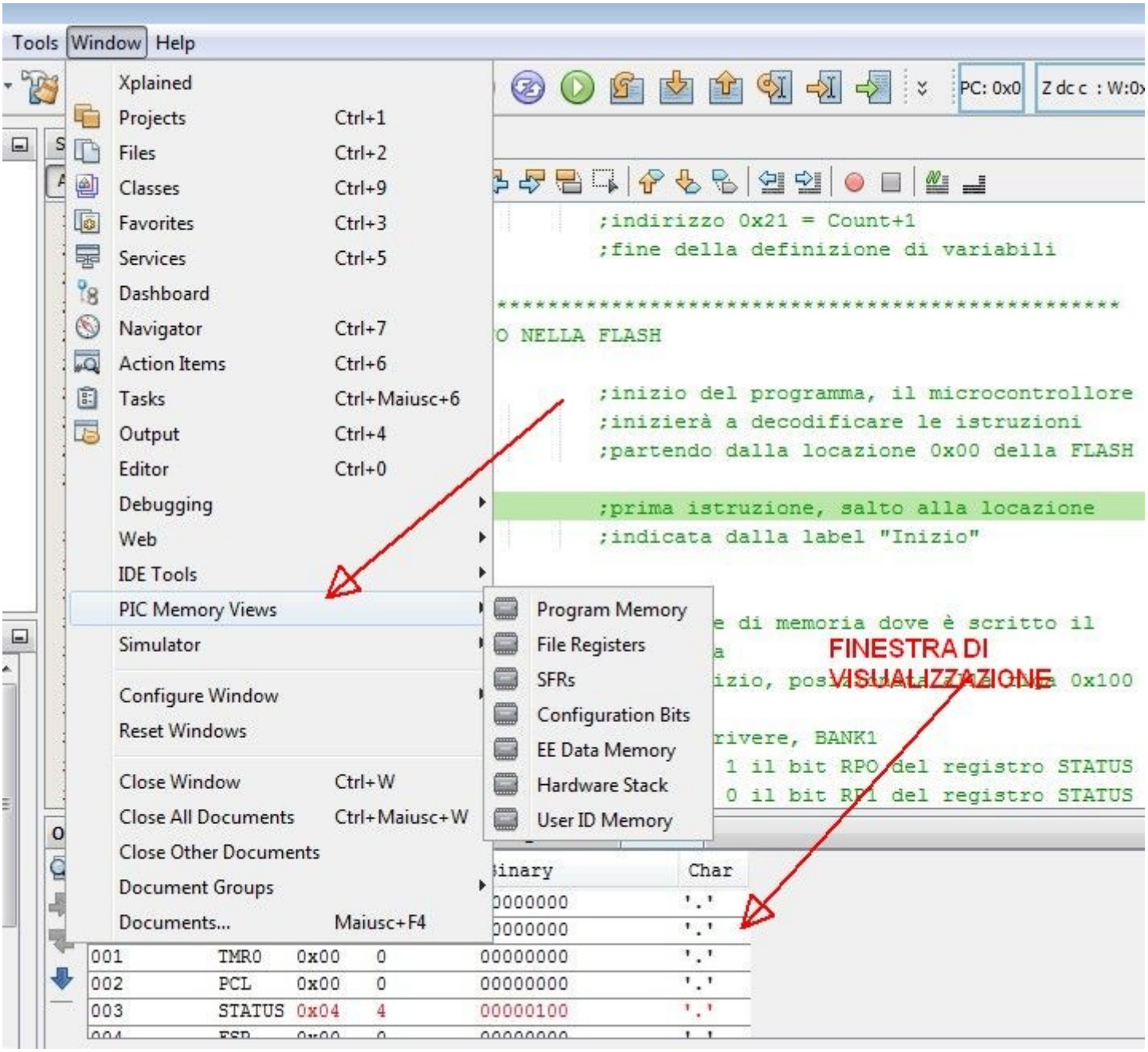
www.danielepostacchini.it

A questo punto possiamo eseguire il programma muovendoci con i tasti di avanzamento a step, in pratica viene eseguita una riga di programma per volta in modo da poter vedere il risultato visualizzando i registri o l'area di memoria del microcontrollore.



www.danielepostacchini.it

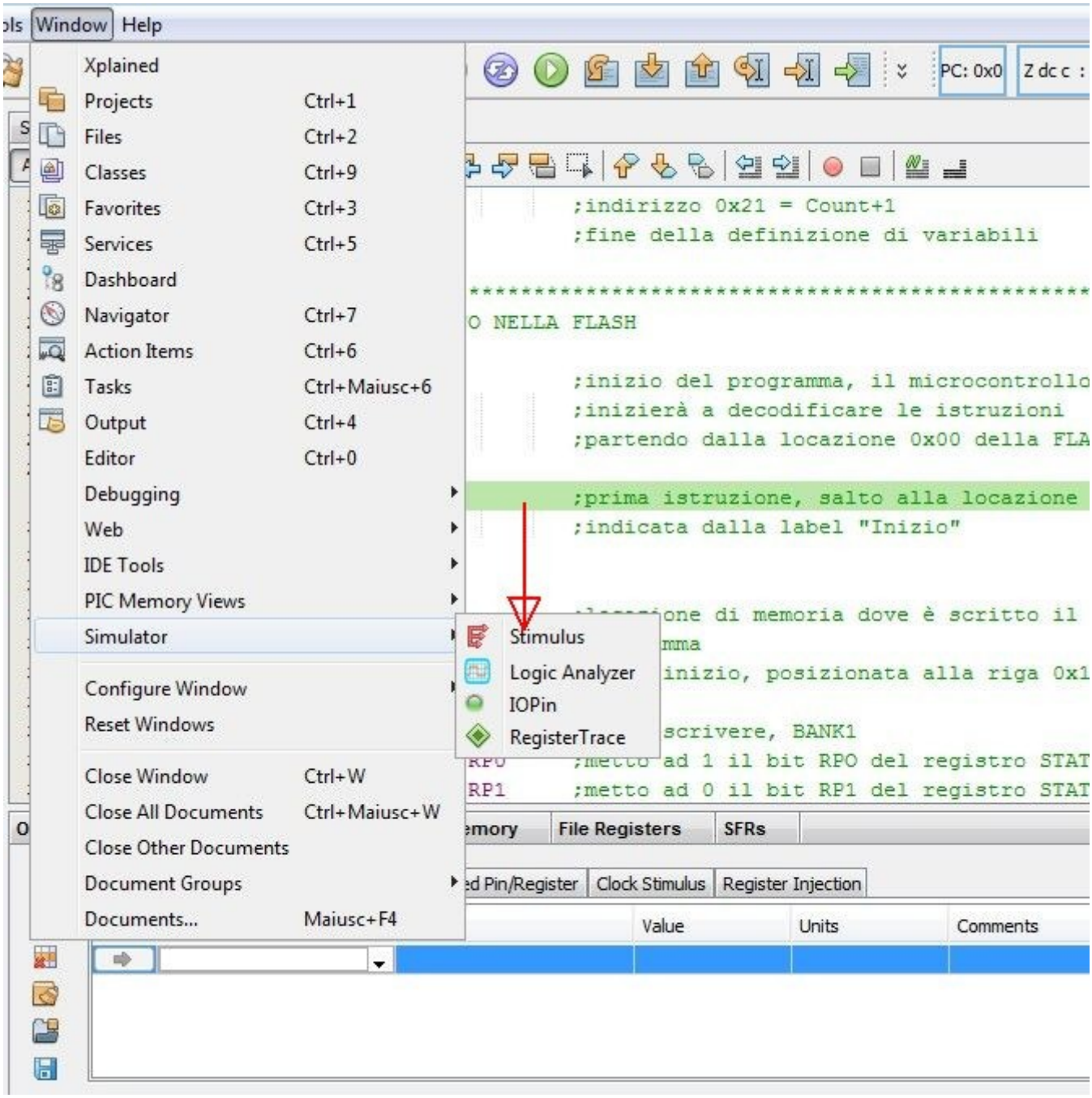
Dal menù Window si potrà scegliere cosa visualizzare nella finestra di visualizzazione come ad esempio i registri (**SFRs**) .



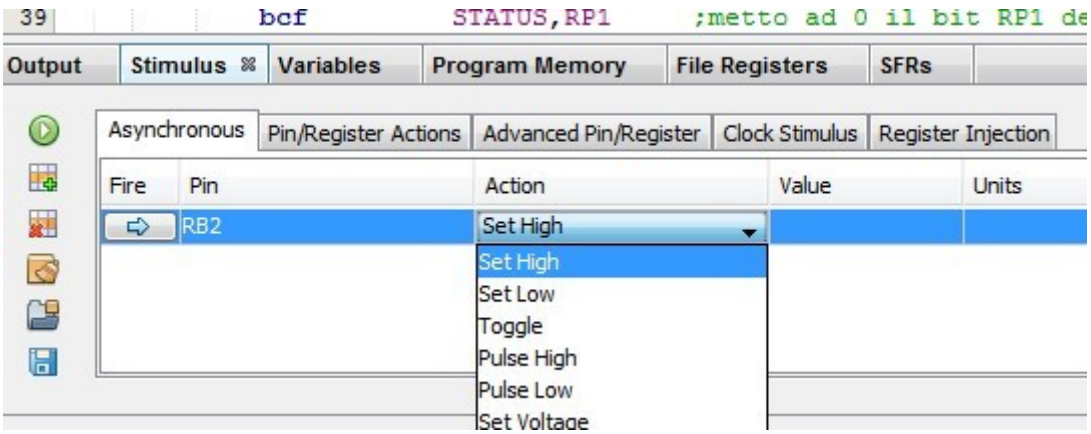
Eseguendo il programma passo passo si possono così controllare lo stato degli ingressi o la variazione delle variabili residenti in memoria.

www.danielepostacchini.it

E' possibie anche simulare l'attivazione di un ingresso tramite il pulsante ***"Stimulus"***



In questo modo nella finestra in basso si possono selezionare dei bit di ingresso scegliendo l'azione tra le varie possibili. In questo modo durante l'esecuzione passo-passo si può attivare o disattivare un ingresso.



www.danielepostacchini.it

Quanto visto finora ci consente di scrivere, compilare e simulare dei semplici programmi in linguaggio assembly. Con molta pazienza si può utilizzare questo linguaggio per realizzare anche programmi più complessi, con i moderni compilatori è possibile però utlizzare linguaggi più evoluti come il basic, o il linguaggio C.

Esistono in commercio diversi ambienti di sviluppo che consentono di sviluppare programmi in linguaggio C utilizzando anche delle librerie già pronte che aiutano a gestire i dispositivi più comuni (display, tastiere ecc...) o i protocolli di comunicazione standard (I2C, USART, RS485, ecc...).

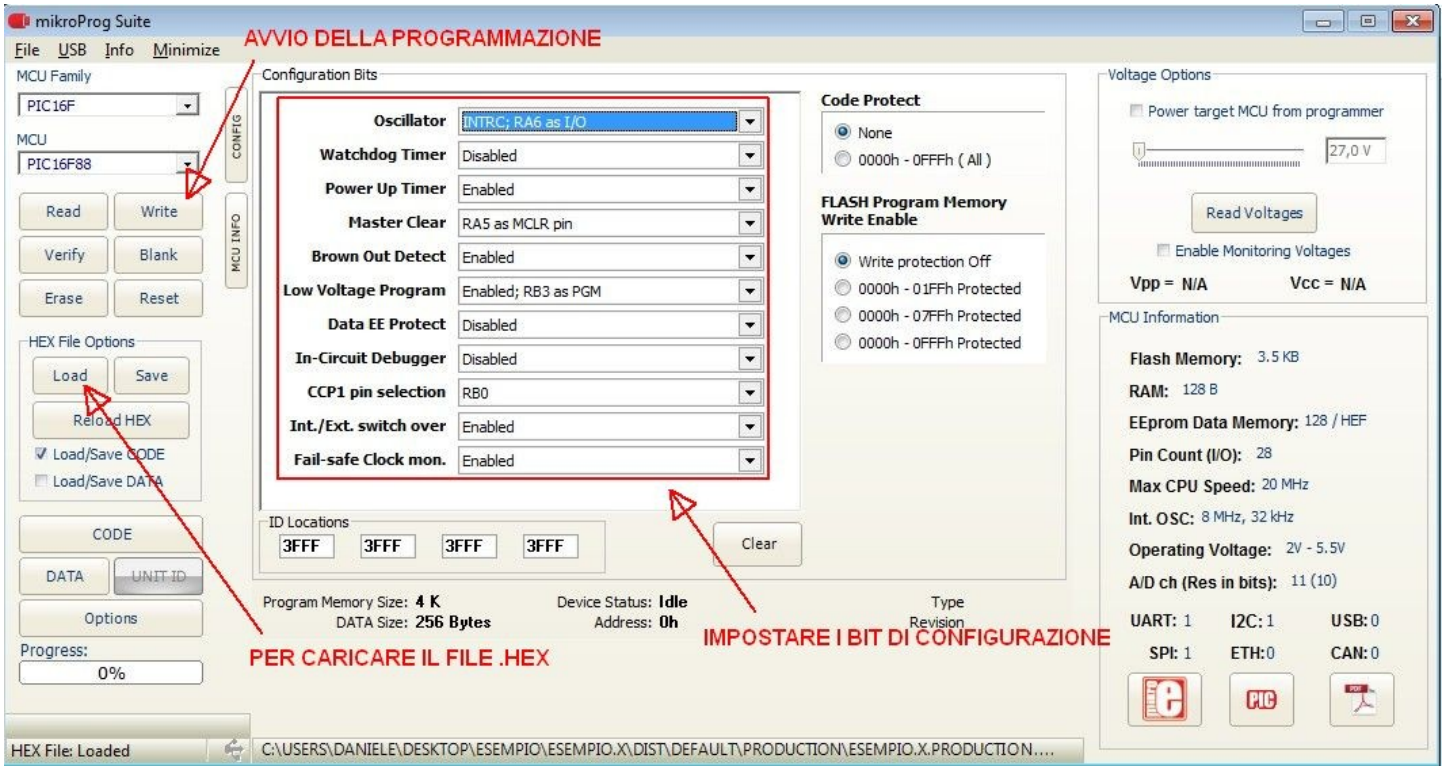
Anche lo stesso MPLAB X consente di sviluppare i programmi in linguaggio C. In questa dispensa useremo invece un compilatore della ditta MIKROELEKTRONIKA (www.mikroe.com) che offre una grande panoramica di software ed hardware per tutti i microcontrollori della microchip.

I compilatori della MIKROELEKTRONIKA, sono inoltre liberamente scaricabili ed utilizzabili per programmi che generano file in linguaggio macchina inferiori a 2KB.

Il primo passo è scaricare l'ultima versione del software MikroC dal seguente link:
https://www.mikroe.com/mikroc-pic

Tramite il file scaricato si installerà l'intero ambiente di sviluppo, compreso di tools di utilità e programmatore.

Lanciando il programmatore ***MikroProg suite for C Pic*** si accederà alla seguente schermata che ci consente di caricare anche il file .hex creato con MPLAB all'interno del PIC. Occorre sempre impostare i bit di configurazione.



Nella prossima dispensa realizzeremo dei programmi con MikroC e vedremo le principali librerie.