**DIRETTIVE**

Partiamo dalla prima :

* **PROCESSOR** è una direttiva del compilatore assembler che consente di definire per quale microprocessore è stato scritto il SORGENTE. Le direttive non sono delle istruzioni mnemoniche che il compilatore traduce nel rispettivo opcode, ma delle semplici indicazioni rivolte al compilatore per determinarne il funzionamento durante la compilazione. In questo caso per esempio

PROCESSOR 16F84

informiamo il compilatore che le istruzioni che abbiamo inserito nel nostro source sono relative ad un PIC16F84.

* **RADIX** serve ad informare il compilatore che i numeri riportati senza notazione, sono da intendersi come numeri decimali. Ovvero se intendiamo specificare, ad esempio il numero esadecimale 10 (16 decimale) non possiamo scrivere solamente 10 perché verrebbe interpretato come 10 decimale, ma 10h oppure 0x10 oppure H'10'. La riga appare così :

RADIX DEC

* **INCLUDE**

#INCLUDE "P16F84.INC"

Con questa direttiva, per esempio, indichiamo al compilatore la nostra intenzione di includere nel source un secondo file denominato “P16F84.INC”. Il compilatore si limiterà a sostituire la linea contenente la direttiva INCLUDE con il contenuto del file indicato e ad effettuare quindi la compilazione come se fosse anch'esso parte del nostro source. Questo tipo di file, in modo particolare, offre la possibilità di utilizzare per esempio il nome dei registri, infatti sostituirà ad ogni nome l’indirizzo della memoria. All’interno ci saranno molte direttive come la seguente.

* **EQU** La direttivaè molto importante in quanto ci consente di definire delle costanti simboliche all'interno del nostro source. Se per esempio si scrive :

LED EQU 0

in particolare la parola **LED** da questo punto in poi del source sarà equivalente al valore 0. Lo scopo principale dell'esistenza della direttiva EQU è quindi rendere i source più leggibili e consentire di cambiare i valori costanti in un unico punto del source. E' importante notare che la parola LED non identifica una variabile ma semplicemente un nome simbolico valido durante la compilazione. Non sarà quindi possibile inserire istruzioni tipo LED = 3 all'interno del source in quanto l'assegnazione dinamica di un valore ad una variabile è un'operazione che richiede l'intervento della CPU del PIC e che quindi deve essere espressa con istruzioni e non con direttive.

* **ORG**

è una direttiva e ci consente di definire l'indirizzo da cui vogliamo che il compilatore inizi ad allocare i dati

o le istruzioni seguenti la riga.

ORG 0CH

In questo caso stiamo per definire un'area dati all'interno del PIC ovvero un'area in cui memorizzare variabili e contatori durante l'esecuzione del nostro programma. Quest'area coincide con l'area RAM del PIC definita dalla Microchip come area dei **FILE REGISTER**. I file register altro non sono che locazioni RAM disponibili per l'utente a partire dall'indirizzo 0CH. Questo indirizzo di inizio è fisso e non può essere cambiato in quanto le locazioni precedenti sono occupate da altri registri specializzati per uso interno. Se nella riga successiva a questa ci sono istruzioni mnemoniche, l’indirizzo della direttiva riguarderà la program memory.

* **RES**

PLUTO RES 2

In questa linea incontriamo una label: **PLUTO** e una direttiva: **RES**.

La direttiva RES indica al compilatore che intendiamo riservare un certo numero di byte o meglio di file register all'interno dell'area dati, in questo caso 2 byte. La label PLUTO, dove PLUTO è un nome scelto da noi, è un marcatore che nel resto del source assumerà il valore dell'indirizzo in cui è stato inserito. Precedentemente deve essere definito l'indirizzo di partenza a 0CH con la direttiva ORG, PLUTO varrà 0CH. Se ad esempio inseriamo una label anche alla linea successiva essa varrà 0CH + 2 (due sono i byte che abbiamo riservato) ovvero 0EH. I nomi delle label possono essere qualsiasi ad eccezione delle parole riservate al compilatore quali sono le istruzioni mnemoniche e le direttive. Una label si distingue da una costante simbolica perchè il suo valore viene calcolato in fase di compilazione e non assegnato da noi staticamente.

* **#DEFINE**

Come nel linguaggio C standard ANSI, consente di definire delle porzioni di testo da associare ad un'unica parola. Se per esempio scriviamo :

#DEFINE ON\_MOT BSF PORTA,0

In fase di compilazione, verrà sostituita la riga con **ON\_MOT** con l’istruzione **BSF PORTA,0**.

* **IFDEF, ENDID, IFNDEF e ELSE**

Serve a isolare eventuali parti di codice da usare solo in certe occasioni, la conseguenza è che non viene compilato. Se devo escludere poche righe lo posso fare con un “;” davanti, se le righe sono molte e magari in varie parti del programma meglio usare questa direttiva. Se per esempio :

**#DEFINE EXTRA**

La direttiva

**#DEFINE** nome parte programma

Attiva la compilazione delle righe all’interno delle due direttive

**IFDEF** nome parte programma

.

.

 **ENDIF**

Istruzioni programma

.

.

Istruzioni programma

**IFDEF EXTRA**

Programma non compilato se presente la direttiva “#DEFINE EXTRA”

.

.

Programma non compilato se presente la direttiva “#DEFINE EXTRA”

**ENDIF**

Istruzioni programma

.

.

Istruzioni programma

Se non voglio compilare questa parte di programma, posso cancellare la direttiva #**DEFINE** oppure mettere un “;” davanti. Se ho più parti di programma da dover escludere o includere li scrivo all’interno delle due direttive mettendo lo stesso “nome parte programma”. Se ho più parti da isolare, con combinazioni diverse, uso dei “nome parte programma” diversi e una direttiva #define per ogni parte diversa.

 Le direttive **IFNDEF** e **ELSE** sono equivalenti, inserite tra le due precedenti permettono di compilare la prima parte se presente la direttiva #**DEFINE**, in alternativa la porte dopo fino a **ENDIF**.

***Le direttive hanno senso solo durante la compilazione del source quindi un PIC non potrà mai eseguire una direttiva.***