Organisation d'un fichier « .asm »

Les commentaires

Tout ce qui suit le symbole « ; » est considéré comme zone de commentaire, vous pouvez y mettre tout ce que vous voudrez.

Les directives

La DIRECTIVE est destinée à MPASM pour indiquer quel type de processeur est utilisé dans ce programme.
Les DIRECTIVES ne font pas partie du programme, elles ne sont pas traduites en OPCODE, elles servent à indiquer à l'assembleur de quelle manière il doit travailler. Ce sont donc des COMMANDES destinées à l'assembleur en lui-même.

Au contraire, les INSTRUCTIONS seront traduites en OPCODE et chargées dans le PIC. Il est donc impératif de bien faire la distinction.

les fichiers « include »

Le Include signale à l'assembleur que les ASSIGNATIONS sont dans le fichier P16F87.inc. Que contient ce fichier ? Et bien tout simplement la valeur de toutes les CONSTANTES que nous allons utiliser.

Pour voir ce qu'il contient, allez dans le menu « file ->Open », choisissez « all source files » dans le cadre inférieur, et ouvrez p16F84.inc. dans le répertoire : C:\MPLAB\IDE\MCHIP_Tools. Une fois dépassée la zone de commentaires, vous verrez des lignes du style :

```
FSR EQU H'0004'
```

Cette ligne signifie tout simplement que FSR est EGAL à 0x0004. Autrement dit, lorsque vous utiliserez FSR dans une instruction, MPASM interprétera FSR comme étant 0x04. 0x04 étant tout simplement l'adresse de FSR dans la mémoire du PIC.
H'0004' est une autre méthode autorisée pour exprimer un nombre hexadécimal, tout comme 04h Si vous prenez votre tableau 4-2 page 13, vous constaterez que c'est bien le cas. Ce fichier est donc principalement destiné à vous éviter d'avoir à mémoriser toutes les adresses, un nom est bien plus simple à utiliser et à retenir. Fermez le fichier p16F84.inc pour ne pas encombrer votre fenêtre.

La directive _CONFIG

La ligne suivante, commence par « __CONFIG ». Cette ligne contient les fameux « fusibles » qui fixent le fonctionnement du PIC.
Les valeurs écrites ici seront intégrées dans le fichier « .hex » pour signaler au programmateur les valeurs à encoder aux adresses spécifiques du PIC.

Sachez donc que si un fichier « .hex » a été créé par un programmeur attentif qui a utilisé cette directive, vous n'aurez nul besoin de définir ces paramètres au moment de la programmation. Le logiciel du programmateur ira normalement chercher ces valeurs dans le fichier lui-même.

Les assignations

A quoi cela sert-il ? Et bien à faciliter la MAINTENANCE de votre programme. Il est en effet plus simple de retenir dans votre programme la valeur « MASQUE » que de manipuler la valeur 0x5B.

Les assignations se comportent comme une simple substitution. Au moment de l'assemblage, chaque fois que l'assembleur va trouver une assignation, il la remplacera automatiquement par sa valeur.
Un autre avantage est que si vous remplacez la valeur d'une assignation, le changement sera effectif pour tout le programme. Vous ne risquez donc pas d'oublier des valeurs en chemin.

Je vous encourage vivement à utiliser les ASSIGNATIONS et autres méthodes que nous allons voir plus bas. La syntaxe est simple puisqu'il s'agit de EQU (égal à)

Exemple d'assignation :
```
mavaleur EQU 0x05
```

Les définitions

Sachez que les « define » fonctionnent comme des ASSIGNATIONS. A ceci près que nous éserverons les assignations pour les valeurs, et les définitions pour remplacer un texte plus omplexe.

Par exemple nous pourrons utiliser un PORT suivi d'un numéro de bit, ou bien carrément une instruction avec ses paramètres.

Une définition est construite de la manière suivante : La directive #DEFINE, suivie par le nom que l'on désire utiliser, puis la chaîne à substituer.

Par exemple :
```
#DEFINE monbit PORTA,1
```
Les macros

```
LIREIN macro
comf PORTB,0
andlw 1
endm
```

La macro se compose d'un nom écrit en première colonne, suivi par la directive « macro ». Commence alors à la ligne suivant la portion de code qui constitue la macro. La fin de la macro est définie par la directive « endm) (end of macro).
Une macro remplace donc un morceau de code que nous utilisons souvent. La macro fonctionne également uniquement comme un simple traitement de texte.

Dans notre exemple, chaque fois que la macro LIREIN sera rencontrée, elle sera remplacée au moment de l'assemblage par les 2 lignes :

```
comf PORTB , 0
andlw 1
```

La macro simplifie donc l'écriture, mais ne raccourci pas la taille du fichier .hex obtenu, puisque les 2 lignes seront écrites dans le PIC.

Notez également que vous disposez d'une aide dans le menu « help->MPASM Help ». En effet, l'aide de MPLAB concerne l'utilisation du logiciel. Les aides concernant le langage sont dans MPASM, puisque c'est ce langage que MPLAB utilise (revoyez l'édition des nœuds).

La zone des variables

Toute zone définie par l'utilisateur commence avec la DIRECTIVE CBLOCK, suivie par l'adresse du début de la zone.
Pour placer nos variables, qui sont des emplacements mémoires auxquels on a donné un nom, nous consultons de nouveau le tableau 4-2. Nous voyons que la zone RAM librement utilisable commence à l'adresse 0x0C pour le 16F84. Notre zone de variable contiendra donc la directive

```
CBLOCK 0x00C ; début de la zone variables
```

Ensuite, vous pouvez utiliser 68 emplacements mémoire, qui répondront à la syntaxe suivante : nom de la variable suivi du signe « : » suivi de la taille utilisée.

Par exemple :

```
w_temp :1 ; Zone de 1 byte
montableau : 8 ; zone de 8 bytes
```

Ensuite, vous devrez préciser la fin de la zone en cours à l'aide de la directive :

```
ENDC ; Fin de la zone
```

Les étiquettes

Vous trouverez dans les programmes en 1ere colonne ce que nous appellerons des ETIQUETTES. Ce sont des noms que vous choisissez et qui sont des REPERES pour le programme.
L'assembleur les remplacera par l'adresse du programme à l'endroit où elles sont positionnées. Ceci vous évite de devoir calculer les emplacements programme. Nous en verrons plus loin le principe.

La directive « ORG »

La directive ORG, suivie de l'adresse, précise à quelle adresse les instructions qui suivent seront placées dans le PIC. Il est important de savoir 2 choses :
Après un reset ou une mise sous tension, le PIC démarre toujours à l'adresse 0x00.
Le début de votre programme doit donc se situer là.
L'adresse 0x04 est l'adresse utilisée par les interruptions (nous verrons le principe plus tard). Il ne vous reste donc pas grand place pour placer votre programme. Nous commencerons donc par un saut vers l'emplacement du programme principal où nous aurons plus de place.

```
org 0x000 ; Adresse de départ après reset
goto init ; Adresse 0: initialiser
```

La première ligne est une DIRECTIVE qui indique que la ligne suivante sera placée à l'adresse 0x00. La seconde ligne est une INSTRUCTION, expliquée page 62, qui indique au PIC que le programme doit SAUTER à l'adresse « init ». « init » est une ETIQUETTE.

Après le reset, le PIC exécute donc l'instruction goto init qui se trouve à l'adresse 0x00, suivie par l'instruction qui se trouve à l'adresse init plus bas dans le programme (donc juste en dessous de l'étiquette init).

La directive « END »

Cette directive précise l'endroit où doit cesser l'assemblage de votre programme. Elle est obligatoire dans tout programme, sous peine d'une erreur qui vous signalera que la fin de fichier (End Of File) a été atteinte avant de rencontrer la directive END.
Toutes les instructions situées après la directive END seront tout simplement ignorées.