CH-1092 Belmont, Switzerland
info@didel.com, www.didel.com

PIC*Génial* - Microcontrôleur PIC 16F84/16F870

Introduction à la programmation avec le PIC*Génial* - suite

8. Pseudos de l'assembleur et séquences "boa"

8.1. Pseudoinstructions

Les pseudo-instructions sont des commandes à l'assembleur. Toutes les pseudo-opérations commencent par un point. Les pseudo-instructions sont composées soit d'une pseudo-opération seulement, soit d'une pseudo-opération suivie par une ou plusieurs expressions (séparées par des virgules). Il existe deux types de pseudo-instructions.

Seules les pseudos CALM qui ont un sens avec les processeurs 8 bits PIC sont documentées par la suite.

8.1.1. .APC

Description: Choisit un des compteurs d'adresse de l'assembleur (APC).

Commentaire: Dans les petits programmes, on n'utilise seulement qu'un seul compteur d'adresse. Avec la pseudo-instruction .APC le programmeur peut choisir entre plusieurs compteurs d'adresses. Jusqu'à huit valeurs différentes d'APC sont usuelles. Toutes les valeurs d'APC sont mis à zéro au début du programme.

Avec le PIC, il peut être utile de définit trois compteurs d'adresse: pour les variables, pour les tables (qui doivent se trouver de préférence en page 0), et pour le programme.

Exemple:

```
        Var     = 0
        Progr   = 1
        Tables  = 2
                .APC    Var         ; Variables à partir de 16'C
                .LOC    16'C
                 .16    C0          ;
                 ... variables globales
                .APC    Progr       ; Programme à partir de 16'100
                .LOC    16'100
                .APC    Tables
                .LOC    16'0
                Jump    Debut       ; Le processeur part en 0
                .Loc    16'4
                Jump    Interruptions
                .APC    Programme
        Debut:
                ... Le programme
                .APC    Tables
                ... Des tables
                .APC    Var
                ... Les variables d'un module
                .APC    Tables
                ... Les tables de ce module
                        ... etc
Tout à la fin du programme
                .APC    Var
                 .IF    APC .HS. 16'3F
                  .Error "Trop de variables"
                .APC    Tables
                 .IF    APC .HS. 16'100
                 .Error "Les tables débordent la page 0"
                .APC    Progr
                 .IF    APC .HS. 16'400
                  .Error "Le programme est trop long pour un 16F84"
        .End
```

8.1.2. .BASE

Description: Définit la nouvelle base par défaut pour les nombres.

Commentaire: L'assembleur CALM démarre avec une base décimale. Les bases courantes sont: binaire, octale, décimale et sexadécimale (hexadécimale). La nouvelle base doit être exprimée dans l'ancienne base sauf quand la base est explicitement placée en tête (p.ex. 16'10).

Exemple:

```
        .BASE   10'16
                Move    #0AB,W
        .BASE   10'2
                Move    #10101011,W
        .BASE   10'10
                Move    171,W
; Chaque fois, la même valeur est chargée dans W
```

8.1.3. .IF

Description: Les lignes d'instructions suivantes jusqu'au .ELSE ou .ENDIF correspondant sont assemblées, si l'expression est vraie (TRUE). Les .IF peuvent être imbriqués.

Commentaire: Cette pseudo-instruction permet l'assemblage conditionnel.

Exemple:

```
        .IF     DEBUG
                ...     ; est assemblé si DEBUG = TRUE
        .ENDIF DEBUG    ...
```

8.1.4. .ELSE

Description: Les lignes d'instructions suivantes jusqu'au .ENDIF correspondant sont assemblées, si l'expression du .IF correspondant était fausse (FALSE). Un texte quelconque suivant .ELSE est ignoré par l'assembleur.

Commentaire: Le texte qui suit la pseudo-instruction .ELSE est seulement une aide supplémentaire pour le programmeur. C'est surtout le cas, quand les pseudo-instructions .IF, .ELSE et .ENDIF sont très éloignées et imbriquées.

8.1.5. .ENDIF

Description: Termine une section d'un .IF ou .ELSE. Un texte quelconque suivant .ENDIF est ignoré par l'assembleur.

Commentaire: Le texte qui suit la pseudo-instruction .ENDIF est seulement une aide supplémentaire pour le programmeur. C'est surtout le cas, quand les pseudo-instructions .IF, .ELSE et .ENDIF sont très éloignées et imbriquées.

Exemple: voir .IF.

8.1.6. .END

Description: Termine les instructions du programme. Des commentaires supplémentaires peuvent se trouver après cette pseudo-instruction.

Commentaire: L'assembleur ignore toutes les lignes d'instructions éventuelles après cette pseudo-instruction. La pseudo-instruction .END peut aussi terminer un fichier inséré (avec la pseudo-instruction .INS).

8.1.7. .ERROR

Description: Génère un message d'erreur.

Exemple:

```
        .IF   APC.and.16'400
           .ERROR  Erreur: le programme déborte la ROM
        .Endif
```

8.1.8. .INS

Description: Insère le fichier mentionné. Le nom du fichier obtient la même extension que le fichier source.

Commentaire: Pour un programme complexe et de grande taille il est souvent plus commode, de diviser le programme en plusieurs fichiers. Le programme principal insère ensuite ces sous-programmes séparés. De plus, on pourrait réunir des constantes souvent utilisées dans un fichier.

Exemple:

```
         .INS      Definitions.ASI
```

8.1.9. .LIST

Description: Les lignes d'instructions suivantes sont copiées dans le listage, si l'expression est vraie (TRUE). Les pseudo-instructions .LIST peuvent être imbriquées.

Commentaire: Cette pseudo-instruction permet le listage conditionnel.

Exemple:

```
.LIST   DEBUG
        ...  ; cette partie du programme apparait si DEBUG = TRUE
.ENDLIST DEBUG
```

8.1.10. .ENDLIST

Description: Termine une section de listage conditionnel.

Commentaire: On peut ajouter un texte après la pseudo-instruction .ENDLIST. Ce texte est ignoré par l'assembleur, mais augmente la lisibilité, si l'on répète l'expression de la pseudo-instruction .LIST correspondante. Ceci est surtout conseillé, si le programme est long et/ou les pseudo-instructions .LIST sont imbriquées.

Exemple: voir .LIST.

8.1.11. .LISTIF

Description: Affiche toutes les pseudo-instructions .IF/.ELSE/.ENDIF dans le listage (.LST).

Commentaire: S'il n'y a pas d'expression ou si l'expression est non nulle, alors l'affichage est fait.

Exemple:

```
.LISTIF DEBUG_IF
```

8.1.12. .LOC

Description: Assigne une nouvelle valeur au compteur d'adresse de l'assembleur courant (APC). Les valeurs de l'APC sont mises à zéro au début du programme.

Exemple:

```
.LOC    16'200
```

8.1.13. .16 Val

Description: Insère la valeur donnée dans le programme objet. Chaque valeur a la taille 14 ou 12 bits selon le PIC.

Exemple:

```
.16     "M","y","p","r","o","g"
Place ces codes dans le programme pour l'identifier
```

8.1.14. .Blk.16 n

Description: Réserve une place de n mots mémoire. Chaque valeur a la taille 14 ou 12 bits selon le PIC.

Exemple:

```
.Loc    16'C
Var1:   .Blk.16  1      ; Variable 1 byte
Tabl:   .Blk.16  4      ; Tableau de 4 variables d'un byte
```

8.1.15. .Fill.16 n,d

Description: Remplit n positions mémoire avec la donnée d de 12 ou 14 bits selon le PIC

Exemple:

```
.Loc    0
.Fill.16  20,0    ; Ecrase les 20 premières positions d'un 12C508 (seront exécutées comme des NOP)
Ceci permet d'écraser un premier programme de test pour en mettre un autre derrière
```

8.1.16. .PROC

Description: Insère la description du processeur mentionné et génère du code machine pour ce processeur.

Commentaire: On peut construire un assembleur CALM de telle manière, qu'il se compose d'une partie principale et d'une partie spécifique pour chaque processeur. Ainsi il n'existe qu'un programme d'assembleur, mais plusieurs descriptions de processeurs. Leurs fichiers correspondants sont chargés avec la pseudo-instruction .PROC.

Exemple:

```
.PROC   16F64
```

8.1.17. .REF

Description: Insère la table des symboles mentionnée. Ainsi les noms et les valeurs des symboles sont définis.

Commentaire: Avec la pseudo-instruction .REF on peut charger des symboles avec des valeurs fixes directement dans la table des symboles de l'assembleur. Ces symboles sont: - des adresses mémoire fixes (p.ex. en ROM) - des définitions d'entrée/sortie - des caractères ASCII non-imprimables (p.ex. CR, LF) Le format de ce fichier dépend de la structure interne de la table des symboles de l'assembleur. Cette

méthode a deux avantages. Premièrement, le chargement de ce fichier est très rapide (l'analyse tombe) et deuxièmement, ce fichier utilise moins de place mémoire par rapport à la méthode équivalente avec un .INS.

Exemple:

```
.REF    Pic16F84        ; Charge le fichier Pic16F84.ref
```

8.1.18. .TITLE

Description: Début d'une nouvelle page avec le titre indiqué dans l'en-tête. Ce titre est répété dans les pages suivantes.

Commentaire: Cette pseudo-instruction se trouve normalement au début d'un programme. On devrait indiquer le nom du programme. Aussi la date et l'heure sont indiqués dans l'en-tête du listage.

Exemple:

```
.TITLE  MonProgramme
```

8.2. Macroinstructions

Les macros sont pratiques pour remplacer des groupes d'instructions répétitifs. Ce qui suit ne documente que leur utilisation simple. Lorsque la macro a été définie, une instruction unique, appelée macro-instruction, copie dans le programme une ou plusieurs instructions d'assembleur et le programmeur est libéré de répétitions embarrassantes. En outre, on peut comprendre une instruction de macro comme instruction d'un langage de haut niveau. Ceci est surtout intéressant pour le programmeur puisque lui-même a conçu ce langage. En abuser réduit la lisibilité par d'autres programmeurs. Avec le PIC, le problème est toutefois que l'on perd le contrôle du temps d'exécution du programme, et la visibilité de l'effet de la macro sur W, sur certaines variables et sur les fanions.

Le macros sont recommandées pour les instructions d'entrée-sortie qui peuvent être amenées à changer de place d'une implémentation à l'autre.

8.2.1. Différence sous-programme - macro

Un sous-programme est caractérisé par le fait qu'il n'y a qu'une copie du texte de sous-programme dans le programme. Le sous-programme est exécuté si l'instruction de programme qui appelle le sous-programme, est exécutée. Les paramètres transmis au sous-programme déterminent sa fonction dans les limites établies au développement du sous-programme. Des sous-programmes diminuent la longueur totale d'un programme ainsi que l'effort pour les écrire puisqu'un appel remplace un sous-programme entier.

Cependant, les économies en mémoire et en effort coûtent du temps, car il faut appeler le sous-programme, passer les paramètres, accèder aux paramètres et quitter le sous-programme. La macro ou "le code de ligne" offre une solution plus rapide aux frais de la place mémoire et de l'effort. Une copie d'une instruction de macro au lieu d'un appel de sous-programme est placée dans le programme. Des modifications ne se font pas au moment de l'exécution mais lors de la programmation et permettent toutes les fonctions désirées par le programmeur. Ainsi on trouve plusieurs copies de textes spécialisés dans le programme au lieu d'une seule copie d'un texte général.

8.2.2. .MACRO nom_de_la_macro

Cette pseudo-instruction commence la définition d'une macro. Le nom de cette macro est nom_de_la_macro. Les caractères admis dans nom_de_la_macro sont les mêmes que pour les étiquettes dans l'assembleur CALM. Actuellement, 32 caractères sont significatifs dans nom_de_la_macro. Une autre macro ne peut pas être définie à l'intérieur de la définition d'une macro. Mais on peut appeler d'autres macros dans une macro, si elles sont définies avant.

Les paramètres de la macro sont appelés dans la partie principale de la macro par les symboles %1 à %8, dans l'ordre des paramètres de la macro-instruction. Les espaces et les tabulateurs sont permis à l'intérieur d'un paramètre.

8.2.3. .ENDMACRO

Cette pseudo-instruction termine la définition d'une macro.

8.2.4. .LOCALMACRO étiquette1,...,étiquette8

Cette pseudo-instruction définit jusqu'à huit étiquettes qui peuvent être utilisées dans la partie principale d'une macro. Ces étiquettes seront converties en étiquettes locales (M_0$..M_999$) si cette macro est appelée. Ceci évite des étiquettes à double si cette macro est appelée plusieurs fois. La syntaxe permise pour les étiquettes est la même que pour les étiquettes dans l'assembleur CALM. Une virgule sépare les étiquettes. Des espaces et des tabulateurs avant et après les étiquettes sont ignorés. Si cette pseudo-instruction est utilisée, elle doit directement suivre .MACRO.

Exemple:

```
.Macro Delai        ; paramètre: délai en multiple de 3µs
.Localmacro  A
        Move     %1,W
        Move     W,C1
  A:    DecSkip,EQ C1
        Jump     A
.EndMacro
```

L'appel se fait par

```
Delai     30        ; 30 * 3µs
```

Un autre exemple mets en macro une comparaison à trois paramètres: les deux opérandes (est-ce que le 2e est plus petit/plus grand que le premier) et l'adresse de saut.

Macros **Super-instructions** **JumpEQ #sig,var,adr**

```
.macro          JumpEQ
      Move      %1,W
      Xor       %2,W
      Skip,NE
      Jump      %3
.endmacro
JumpNE #sig,var,adr
.macro          JumpNE
      Move      %1,W
      Xor       %2,W
      Skip,EQ
      Jump      %3
.endmacro
```

```
; JumpLO Var/Cste, Var, Adresse de saut si Var < Const
.Macro JumpLO
      Move      %1,W
      Sub       W,%2,W        ; %2 + (256-W) -->
      Skip,CS
      Jump      %3
.Endmacro
; JumpHS Var/Const, Var, Adresse de saut si Var >= Const
.Macro JumpHS
      Move      %1,W
      Sub       W,%2,W        ; %2 + (256-W) -->
      Skip,CC
      Jump      %3
.Endmacro
```

Comme exemple d'utilisation, si on attends des caractères par une ligne série, et on veut ignorer les caractères CR et LF, on peut écrire:

```
RecNext:
        Call      RecSer    ; Résultat dans DataRec
; Caractère arrivé; on ignore les CR LF
        JumpEQ    #CR,DataRec,RecNext
        JumpEQ    #LF,DataRec,RecNext
```

8.2.5. .LAYOUTMACRO Comment

Les commentaires d'une macro sont par défaut supprimés. Si on veut les conserver, il faut mettre au début du programme

```
.Layoutmacro   Comment
```

8.2.6. .MACRO avec paramètres par défaut

On peut en déclarant une macro donner une liste de paramètres correspondant aux valeurs par défaut.

.MACRO nom_de_la_macro,Param1,Param2, ...

Par exemple, la Macro "Copy" peut par défaut copier le PortA sur le PortB, et si on précise le paramètre source ou destination, en tenir compte.

```
.Macro Copy,PortA,PortB
Move     %1,W
Move     W,%2
.EndMacro
```

Cette macro n'est en fait pas conseillée car elle ne facilite pas la relecture du programme. Par exemple, elle permet des paramètres par défaut, ce qui est amusant, mais génère facilement des erreurs, et ne facilite pas la compréhension.

```
Copy            génère    Move    PortA,W
                          Move    W,PortB
Copy    Var1    génère    Move    Var1,W
                          Move    W,PortB
Copy    ,Var2   génère    Move    PortA,W
                          Move    W,Var2
Copy    V1,V2   génère    Move    V1,W
                          Move    W,V2
```

8.3. Commandes BOA de mise en page et d'impression

Avec l'éditeur Smile NG la touche F8 passe du mode interprété ou non. Dans le menu "Editor options" "Other" il est possible de dire si les commentaires longs sont tronqués ou passent à la ligne suivante.

Mode non interprété

\titre;xxxx \title;xxxx

\prog;xxxx|yyyy programme

\module;xxxx|yyyy module

\rout;xxxx|yyyy routines

\const;xxxx|yyyy constantes

\var;xxxx|yyyy variables

\macro;xxxx|yyyy macros

\table;xxxx|yyyy table

\b;xxx en gras

\in;xxx paramètres en entrée

\out;xxx paramètres en sortie

\mod;xxx variables modifiées

\iom;xxx in/out/mod 2e ligne

Mode interprété

XXXX

Program **XXXX** yyyy

Module XXXX yyyy

Routine XXXX yyyy

Constant XXXX yyyy

Variables XXXX yyyy

Macros XXXX yyyy

Table **XXXX** yyyy

\b;xxx

in: *xxx*
out: *xxx*
mod: *xxx*
xxx