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



# PIC*Génial* – Microcontrôleur PIC 16F870

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

## 2. Programmation du PIC 16F870

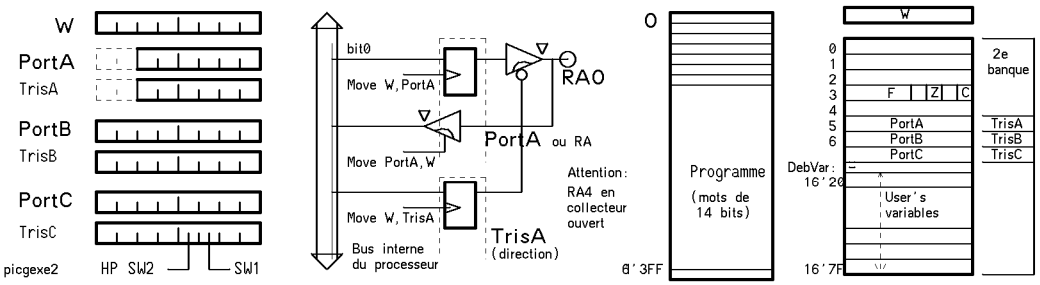
L´architecture, le répertoire d´instruction et les périphériques internes du PIC seront expliquées en plusieurs fois, pour éviter des longs commentaires et des listes rébarbatives d´instructions.

### 2.1. Un premier modèle simplifié du PIC

Le PIC 16F870 peut être vu en première approche comme un registre W, trois ports A, B et C, leurs registres de direction associés. Les port A, B, C sont aux adresses 5, 6, 7 (l´assembleur le sait, il n´y a pas besoin de le lui dire). Les registres de direction sont dans une 2e banque, mais peuvent être accédés directement. Avec CALM, les registres de direction s´apellent TrisA, TrisB et TrisC et l´instruction Move permet de transférer la valeur préparée dans W. On ne peut par contre pas lire le contenu ou faire des opérations comme avec les autres registres et variables. Un 0 dans un bit de direction met la ligne correspondante en sortie (truc mnémotechnique: 0 -> O out 1 -> I In).

Lorsque le processeur exécute par exemple l´instruction "Move W,PortA", une impulsion charge le registre appelé PortA ou RA (register A). Les sorties qui ont été préparées en sortie copient cette information. W --> registre interne --> sorties. Si on relit avec un "Move PortA,W", on lit directement l´état des broches. Le passeur qui met en communication la broche avec le bus interne, lui-même relié au registre W, n´est activé que pendant quelques dizaines de nanosecondes. C´est une photo instantanée de l´état des broches du port; si la tension sur une broche est inférieure à 0.7V, le fabricant garantit qu´un zéro sera enregistré dans W. Si c´est supérieur à 2V (alimentation 5V), c´est un 1. Entre les deux, ce sera 0 ou 1. La frontière dépend du circuit et de sa température.

A partir de la position DebVar=16´20, l´utilisateur peut placer ses variables, compteurs, bits d´état, etc. Il faut passer par le registre W (Work register) pour initialiser une variable à une valeur différente de zéro. Le résultat des opérations entre W et un registre n´est pas nécessairement dans W; la destination peut être le registre, ce qui est souvent très efficace.




picgexe2 *Fig. 1 Modèle pour les ports et les 2 mémoires du PIC*

Le registre F ou STATUS (flags, état) mémorise en particulier deux bits que l´on trouve dans tous les processeurs: C est le "Carry", activé par une addition avec dépassement de capacité, ou un décalage. Z est le "zéro bit", activé si le résultat d´une opération est nul (les 8 bits du résultat transféré dans W ou dans un registre sont nuls). Toutes les instructions n´agissent pas sur Z et sur C: la feuille de codage précise pour chaque intruction les bits modifiés.

Le programme est dans une mémoire séparée des variables (architecture "Harvard"). Le processeur démarre en 0 (nous parlerons des interruptions plus tard) et exécute chaque instruction en 1 microseconde (à 4MHz), sauf les sauts et instructions qui modifient le registre PCL qui "cassent le pipeline" et demandent deux microsecondes.

Un programme commencera toujours par une initialisation des registres de direction des ports B et C. Le registre A sera ignoré dans les premiers exercices. Si on ne fait rien, les ports sont en entrée après un Reset. Il faudra aussi initialiser les variables, compteurs dont la valeur à l'enclenchement est importante. On donnera un nom aux variables, en déclarant au début les équivalences entre ces noms et les adresses mémoire assignées (en évitant de mettre deux variables différentes dans la même adresse mémoire). Les noms des registres du PIC sont connus par l'assembleur ou sont défini dans le fichier 16F870.ref.

## 2.2. Un premier programme

Nous voici enfin à pied d'oeuvre pour devenir des experts dans la programmation du PIC. Comme pour tous les processeurs, il faudrait déjà connaître tout avant de pouvoir expliquer clairement la moindre instruction. Avec des exemples, nous allons progressivement nous familiariser avec l'architecture et le répertoire d'instruction. Toute l'information utile est résumée dans une feuille de codage qui deviendra notre instrument de travail. La documentation complète du fabricant est essentielle pour comprendre toutes les possibilités du processeur (timer, interruptions), mais en première étape on ne peut que s'y perdre. Le document en anglais "Programming the Microchip-PIC microcontrollers" (www.didel.com/picg/doc/PicSoft.pdf) détaille en anglais les instructions en notation CALM et Microchip-PIC, et donne plusieurs exemples de programmes avancés..html

Chargeons et exécutons le programme PicgT; il a l'extension .ASM comme tous les programmes en assembleur, mais cette extension n'a pas besoin d'être précisée. Il copie le port C, initialisé en entrée, sur le port B, initialisé en sortie. Si on presse sur l'un des deux poussoirs du module PicgExe, l'effet sera visible sur deux LEDs. Pouvez-vous prévoir lesquelles? Ne soyez pas sutpis de voir des LEDs qui s'allument et s'éteignent quand vous touchez la carte. Les entrées inutilisées du PortC sont en l'air, à haute impédance, et les influences électrostatiques peuvent changer l'état. Ce n'est pas dangereux, car le PIC a des diodes de protection, mais dans des pays secs, ou avec certains tapis, il faudrait mieux protéger.

```
Program PicgT   Copie le port C sur le port B
.Proc     16F870
.Ref      16F870

Constant Ports  Ports C et B
DirC      = 2'11111111   ; Tout le port A en entrée
DirB      = 2'00000000   ; Tout en sortie
ToutEteint = 2'11111111
.Loc      0

Program Initialisation
Debut:    Move     #DirC,W
          Move     W,TrisC
          Move     #DirB,W
          Move     W,TrisB
          Move     #ToutEteint,W
          Move     W,PortB

Program Boucle  On boucle sans cesse pour copier RC dans RB
Boucle:
          Move     PortC,W
          Move     W,PortB

          Jump     Boucle
.Align    8
.16       "P","i","c","g","t"
.Fill.16  FinProg+1-APC,-1
.Loc      16'2007
.16       16'3F39 ; Config
.End
```

Reprenons ce programme instruction par instruction pour comprendre la raison et le sens de chaque instruction. Ce programme est traduit par l'assembleur en codes binaires qui sont chargées en mémoire. On peut voir les codes générés en chargeant le fichier PicgT.lst, sauvé par l'assembleur dans le même répertoire.

\prog;PicgT|Copie le port C sur le port B

Dans SmileNG et Picolo il y a, en plus des instructions pour le processeur, des ordres de mise en page et des pseudo-instructions pour l'assembleur. Pour avoir sur l'écran un joli graphisme en début de programme, il faut taper la séquence \prog;xx|yy.

Cet ordre est interprété par l'éditeur et par le programme d'impression. Il ne perturbe pas les autres outils d'édition, de transfert et d'impression. C'est donc une solution simple et compatible. Les ordres "LILA" interprétés pas Smile NG sont donnés dans l'annexe 3. Le nom du programme, PicgT est naturellement le même que celui du fichier sur disque dans le répertoire PicGénial: PicgT.asm. L'extension .asm est automatique. Les minuscules et majuscules ne sont pas distinguées.

```
.Proc    16F870
.Ref     16F870
```

Ces pseudo-instructions signalent à l'assembleur le processeur utilisé (16F870 et tous les processeurs de la famille 16F87x qui ont le même jeu d'instructions). Le fichier 16F870.pro doit exister sur le disque, dans le répertoire "Proc" de SmileNG ou Picolo, le fichier 16F870.ref doit se trouver dans le répertoire Ref, et l'assembleur Ascalmc.exe dans le répertoire "Exe".

```
\const;Ports|Ports B et C
```

Un ordre de mise en page que l'assembleur ignore

```
DirC     = 2'11111111   ; Tout le port C est en entrée
DirB     = 2'00000000   ; Tout le port B est en sortie

ToutEteint       = 2'11111111   ; Lampes éteintes
```

Les constantes sont déclarées au début du programme. Ici le port C est voulu en entrée. Les 8 bits de direction doivent être mis à un. 2' signifie que l'on travaille en binaire, ce qui est naturel ici. Les 8 bits du port B sont en sortie. Les bits de direction devront être mis à zéro. Ces pseudo-instructions de déclaration n'ont pas d'effet sur le processeur. Avec ces déclarations en début de programme, on établit un dictionnaire entre un langage qui nous est naturel (DirC pour la configuration de direction du port C) et le langage du processeur, formé de bits seulement.

On voit dans le schéma qu'il faut que le bit en sortie soit à zéro pour que la diode correspondante s'allume. Tous les bits du port B doivent être à un pour que les diodes soient éteintes. La constante "ToutEteint" qui permettra de s'assurer que toutes les diodes sont éteintes à la fin de l'initialisation, est donc un mot avec des "1" partout.

```
\prog:Initialisation|
Debut:   Move     #DirC,W
         Move     W,TrisC
         Move     #DirB,W
         Move     W,TrisB
```

Ce sont les 4 premières instructions exécutées, à partir de l'adresse 0. Elles initialisent la direction des ports. TrisA et TrisB sont les registres de direction. Il faut les initialiser avec les valeurs DirC et DirB, ce qui implique de copier la valeur à transférer dans le registre de travail W (workspace register). On pourrait écrire Move #2'11111,W et ne pas déclarer DirC. Mais c'est prendre une mauvaise habitude que de ne pas déclarer les constantes et variables. Pour les programmes simples, cela simplifie un peu, mais dès que les programmes sont compliqués, on s'empêtre, fait des fautes, et un programme que l'on a écrit soi-même est difficile à comprendre après quelques jours.

Le signe # (lu "valeur" plutôt que "dièze") montre que DirC est une valeur immédiate, fixe dans le programme. La valeur peut être donnée en décimal, en binaire (2'01101) ou en hexadécimal (16'F3); l'assembleur traduira en binaire pour mettre en mémoire programme la valeur que le processeur comprend. TrisC par contre est un registre, dont le contenu est variable. W également. Move W,TrisC transfère le contenu de W dans le registre TrisC. W n'est pas modifié.

A noter encore qu'il n'y a pas de commentaires pour expliquer ces 4 instructions. Leur lecture est évidente. Ce qui est important, c'est que DirC et DirB soient bien expliqués au début du programme, avec tous les détails nécessaires pour que l'on puisse câbler le système conformément au programme.

```
         Move     #ToutEteint,W
         Move     W,PortB
```

Les bits en sortie doivent être assignés, autrement l'état initial sera quelconque (en fait le PIC initialise ses registres à zéro à la remise à zéro (Raz). Ici, on veut que les diodes soient éteintes à l'enclenchement. La réflexion pour savoir s'il faut mettre des 1 ou 0 a été faite au début, dans les déclarations. Il n'y a plus besoin de se reposer des questions. Si on change de système et qu'il faut un 0 pour éteindre, il suffit de changer les déclarations, et rien dans le programme.

```
\prog;Boucle|On boucle sans cesse pour copier RC dans RB
Boucle:
```

L'assembleur note l'adresse mémoire de boucle. Plus loin, l'instruction "Jump Boucle" permet de revenir ici pour répéter l'opération

```
         Move     PortC,W
         Move     W,PortB
```

Ces deux instructions copient le port C (qui comporte en particulier deux poussoirs sur les entrées RC0 et RC1) sur le port B. La copie se fait sur les bits de même numéro.

```
         Jump     Boucle
```

On recommence sans cesse; quelle est la durée de la boucle?

```
.Align   8
.16      "P","i","c","g","t"
.Fill.16 FinProg+1-APC,-1
.Loc     16'2007
.16      16'3F39  ; Config
```

Humm, pas d'explications pour le moment; ces instructions sont liées au programme IC-prog et n'ont rien avoir avec l'exécutions du programme dans le PIC. Bornons-nous pour le moment à les mettre à la fin de chaque programme sans bien comprendre.

.End

La pseudo-instruction .END dit à l´assembleur que son travail de traduction et terminé. Les lignes suivantes sont ignorées. Cela peut être le mode d´emploi de votre programme. Les fichiers se perdent moins facilement que les feuilles de papier. Documentez ce que vous faites autant que possible dans votre programme et pas ailleurs. Après le .End, il n´y a plus besoin de ; devant les lignes, et les lignes peuvent avoir plus de 96 caractères.

S´il n´y a pas d´erreurs, l´assembleur (touche F7) génère le code binaire, sauvé dans le format binaire (PicgT.bin) ou hexadécimal (PicgT.hex) selon la sélection.

Le programme ICprog est chargé et configuré pour programmer le 16F870 (ou 16F870A, ce n´est pas la même procédure). Le fichier .bin ou .hex est chargé et la programmation st lancée. L´exécution démarre en fin de programmation.

A l´exécution, le processeur va initialiser les ports, et ensuite exécuter sans cesse la boucle qui copie le port C sur le port B. La durée de la boucle est 4 microsecondes. Si le poussoir est pressé, l´entrée sur le port C passe à zéro. Ce zéro est copié sur le bit correspondant du port B, et allume la lampe correspondante. Puisque les poussoirs sont sur les lignes RC0 et RC1 du processeur (port A bits 3 et 4), ce sont les LEDs 3 et 4 (donc 4e et 5e depuis la gauche) qui vont être activées. Comme un zéro est actif à la fois pour les poussoirs et pour les LEDs, l´effet est correct. Si on veut que les LEDs non concernées ne s´allument pas, il faut forcer les autres lignes du port B à un, ce qui se prépare dans W avec un OU logique. Il faut insérer entre la lecture du port A et l´écriture sur le port B l´instruction "Or #2´11100111,W". Ceci sera expliqué avec plus de détails plus loin, mais il faut dès maintenant s´habituer à compter les positions de bits correctement (de 0 à 7 depuis la gauche même si on dit 1er au 8e).

## 2.3. Un peu plus sur l´architecture du PIC

Le PIC 16F870 contient en plus de ses trois ports A, B etC, et leurs registres de direction associés, des registres de commande et de mode que nous découvrirons petit à petit. Ils utilisent les adresses 0 à 16´1F sur le 16F870. Une zone de registres est prévue pour les variables à partir de l´adresse DebVar=16´20 (définie dans le fichier 16F870.ref). Il y a deux façons de déclarer les adresses des variables. On peut les assigner avec des déclarations Var1= 16´20 (mieux: DebVar+0), Var2 = 16´21 (DebVar+1), ce que nous ferons dans les premiers programmes simples.

Les bits Z, C et D, réunis dans un registre de "fanions" (flag register F ou STATUS) sont très importants et nous donneront un peu de fil à retordre. Le bit Z est activé par quelques instructions pour signaler que le résultat de l´opération est nul. Par exemple l´instruction "XOR Var,W" calcule le ou exclusif de la variable Var et du contenu du registre W. Z est activé si le résultat est nul, ce qui veut dire que Var et W avaient le même contenu (après exécution de l´instruction la variable Var est inchangée et W vaut zéro dans ce cas). "Move W,Var" ne modifie pas Z, mais "Move Var,W" modifie Z (Z=1 si Var contient zéro). Test Var est en fait un Move Var,Var, et modifie Z. Il faut, au début, regarder chaque fois la feuille de codage (voir plus loin) pour vérifier ce que fait exactement l´instruction.

Le bit C est activé s´il y a dépassement de capacité dans une addition, soustraction ou décalage. Le bit D n´est activé que par l´addition et la soustraction; il est utile pour calculer en décimal, mais il est rarement utilisé.

## 2.4. Initialiser les ports

Comme on l´a vu, tout programme commence par des déclarations: quel est le processeur, quels noms sont donnés aux bits des ports, aux variables. Au début du programme il faut initialiser les directions de ports, en entrée ou en sortie. Avec les PIC, des instructions spéciales permettent d´initialiser ces registres de direction TrisC et TRisB. En CALM, elles s´écrivent Move W,TrisC, mais Move TrisC,W ou Set TrisC:#bit n´est pas possible. Par contre, Move PortC:W et Set PortC:#bit est possible. Un zéro programme une sortie. Par défaut (si on oublie de définir les registres TrisC ou TrisB), les ports sont en entrée après une Raz (remise à zéro), ce qui est logique puisque l´on veut éviter que le processeur entre en conflit avec des signaux allant vers le processeur. Un reset ne modifie

pas les ports A,B,C. Pour initialiser un registre, une variable, il faut toujours avec le PIC passer par le registre de travail W. Par exemple, on initialise les 4 bits de poids faible du port B en sortie avec les instructions:

```
Move    #2'11110000,W    ; 2' pour les nombres binaire, 2'11110000 = 16'F0
Move    W,TrisB
```

Rappelons que le signe # (prononcé dièze ou valeur devant un nom) indique que la valeur est immédiate (paramètre d'assemblage). Il serait préférable de déclarer au début du programme "DirB = 2'11110000" et d'écrire dans le programme "Move #DirB,W". On aime bien avoir toutes les définitions de constantes au début, et il faut toujours nommer et déclarer au début du programme les constantes que l'on peut être amené à modifier par la suite.

Si on veut mettre tous les ports en sortie, et les initialiser à zéro, l'instruction Clr (Clear) existe, mais elle ne peut pas agir sur les registres de direction TrisA et TrisB. On doit donc écrire:

```
Clr     W
Move    W,TrisC
Move    W,TrisB
Clr     PortC           ; ou Move  W,PortC car W contient 0
Clr     PortB           ; ou Move  W,PortB
```

Il est plus propre (les modifications ultérieures seront facilitées) de définir l'état initial et d'utiliser toujours la séquences d'initialisation complète:

```
Move    #DirC,W
Move    W,TrisC
Move    #DirB,W
Move    W,TrisB
Move    #InitC,W
Move    W,PortC
Move    #InitB,W
Move    W,PortB
```

## 2.5. Copie de bits

Nous avons jusqu'à présent manipulé les 8 bits des port B ou C simultanément. Le PIC est très performant pour agir directement sur l'un des bits d'un port ou d'une variable en mémoire (mais pas sur les registres de direction si on les accède par les instructions Tris), sans modifier les autres bits. Les bits sont numérotés de 0 (poids faible) à 7 (poids fort). Le signe : indique une sous-adresse; PortC:#3 est le bit 3 du port C. On nomme naturellement les numéros de bits selon leur fonction. Il y a deux instructions de modification de bits:

```
Clr     PortC:#BitNumber  ; Met le bit à zéro
Set     PortC:#BitNumber  ; Met le bit à un
```

On peut de même lire (en fait tester) un bit d'un port initialisé en entrée, sur lequel on a câblé un interrupteur par exemple, ou un bit dans une variable. Tester un bit veut dire que l'on va prendre une décision. Tout ce que sait faire le PIC est de sauter conditionnellement l'instruction suivante. Le nom de l'instruction est TestSkip et la condition est Bit Set ou Bit Clear (BS, BC). L'adresse du registre ou port suit avec comme sous-adresse le numéro du bit (de 0 à 7), qui est une valeur immédiate, donc avec le signe #.

```
TestSkip,BC  PortC:#bNumber ; Skip si le bit testé est à zéro
TestSkip,BS  PortC:#bNumber ; Skip si le bit testé est à un
```

Par exemple, si on veut copier l'état du poussoir câblé sur RC0 sur le bit de poids fort de l'affichage, RB7, on initialisera au moins RC0 en entrée et RB7 en sortie avant de tester et activer les bons bits. Le programme s'écrit

```
Program Picgt3   Lit et copie un interrupteur
.Proc     16F870

Constant Port C
bPoussoir = 0       ; poussoir en RC0, actif à zéro
DirC      = 2'11111111     ; RC7..0 en entrée

Constant Port B
bLed      = 7       ; Led en RB7
DirB      = 2'00000000     ; Tout en sortie
ToutEteint = 2'11111111
```

```
Program Début du programme
.Loc      0
Debut:    Move      #DirC,W
          Move      W,TrisC
          Move      #DirB,W
          Move      W,TrisB
          Move      #ToutEteint,W
          Move      W,PortB
Boucle:
          TestSkip,BS   PortC:#bPoussoir   ; Si 0 (bit cl
          Clr       PortB:#bLed
          TestSkip,BC   PortC:#bPoussoir   ; dans l'autre
          Set       PortB:#bLed
;... rien d'autre à faire dans ce programme
          Jump      Boucle
.End
```

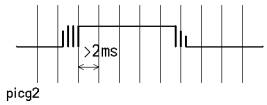
Notons que c'est assez pratique de mettre un b minuscule comme première lettre lorsqu'il s'agit d'un bit et non pas d'un mot de 8 bits. Il ne faut pas avoir peur de passer du temps à choisir et taper des noms explicites; ce temps reste négligeable vis-à-vis du temps passé à trouver des erreurs de programmation.

## 2.6. Supression de rebonds

Nous voulons compter les actions sur le poussoir. Le premier problème est de suivre un signal qui passe à un, puis passe à zéro, et d'exécuter une action (compter et afficher le résultat) pour l'une de ces transitions. Il suffit d'écrire

```
A$:       TestSkip,BS   PortC:#bPoussoir
          Jump    A$        ; Exécuté si bit clear (on attend que le signal monte)
B$:       TestSkip,BC   PortC:#bPoussoir
          Jump    B$        ; Exécuté si bit set (on attend que le signal redescende)
          ; Tâche à exécuter
```

Le deuxième problème à résoudre est que notre poussoir a des rebonds de contact; la lame du contact rebondit pendant quelques millisecondes et le processeur est assez rapide pour voir ces contacts successifs. Une solution est de ne pas lire trop souvent, en insérant une boucle d'attente de 2 ms au moins (et 50ms au plus pour ne pas rater une action très rapide sur le poussoir). Le programme complet appelle une routine d'attente dans laquelle le délai en pas de 100 microsecondes est un paramètre. Il est facile de le réduire pour voir quelle est la durée maximale des rebonds.




picg2 *Fig. 2 Echantillonage pour supprimer les rebonds*

Puisque les diodes sont allumées par un zéro, il faut inverser la valeur du compteur avant de la transférer vers le port B. Le PIC sait faire cela en une seule instruction. L'instruction NOT inverse tous les bits d'un registre. Avec le PIC, "NOT Compteur" inverse tous les bits de la variable compteur, ce qui n'est pas souhaité ici. "NOT Compteur,W" ne modifie pas Compteur, car l'inversion est faite au moment de la copie dans W.

Le programme suivant a besoin de 3 variables. C0 C1 sont des compteurs pour boucle d'attente. Compteur est la variable qui compte les actions sur le poussoir et est copiée sur le port B pour être visualisée à chaque changement. On pourrait déclarer

```
Compteur  = 16'20
C0        = 16'21  ; ou Compteur + 1
C1        = 16'22
```

On préfère comme dans le programme ci-dessous, réserver un position mémoire avec la pseudo-instruction .16 1 (la variable est 8 bits, mais l'assembleur PIC doit considèrer que les variables ont la même taille que les instructions 12, 14 ou 16 bits des processeurs de la famille PIC). Il faut naturellement dire alors à partir de quelle adresse on trouve ce bloc de variables (.Loc DebVar) et mettre au début du programme un .Loc 0 pour dire que le programme commence en zéro.

```
Program PicgT0   Comptage des actions sur le poussoir gauche
.Proc     16F870
.Ref      16F870

Constant Ports
bPoussoir  = 0     ; sur RC0, actif à zéro
DirC       = 2'00000011    ; RC1 RC0 en entrée
DirB       = 0             ; tout en sortie, LEDs actives

Variables Variables
.Loc      DebVar
Compteur: .16       1
C1:       .16       1
C2:       .16       1
.Loc      0

Program Début
Debut:    Move      #DirC,W
          Move      W,TrisC
          Move      #DirB,W
          Move      W,TrisB
          Clr       Compteur
          Not       Compteur,W
          Move      W,PortB
Boucle:
  A$:     Call      Delai
          TestSkip,BC   PortC:#bPoussoir
          Jump      A$    ; Attend que l'on presse
  B$:     Call      Delai
          TestSkip,BS   PortC:#bPoussoir
          Jump      B$    ; Attend que l'on relâche
          Inc       Compteur
          Not       Compteur,W
          Move      W,PortB
          Jump      Boucle

Routine Delai  Delai multiple de 100µs
  in:  W delai 0, 0,1 ... 25,5 ms
  mod: C1 C2 W
Delai:    Move      #20,W   ; Exemple avec 2 ms
          Move      W,C1
  A$:     Move      #32,W   ;Boucle interne 100µs
          Move      W,C2
  B$:     DecSkip,EQ C2
          Jump      B$
          DecSkip,EQ C1
          Jump      A$
          Ret
.End
```

Les routines, avec ses instructions Call et Ret seront expliquées dans le chapitre suivant.

Sauriez-vous remettre le compteur à zéro avec l'autre poussoir? C'est plus simple, car il n'y a pas besoin de se préoccuper des rebonds. Si le poussoir n'est pas activé on passe par dessus une instruction "Clr Compteur" à rajouter. Cette instruction doit être mise dans la boucle qui attend que l'on presse sur le poussoir de comptage. Il faut aussi faire la mise à jour du port B immédiatement, car c'est lui que l'on voit. C'est un peu plus compliqué, et il y a plusieurs solutions. A vous de trouver la plus élégante.

A noter encore que ce genre de programme qui attend sur une touche pressée ou relâchée n'est pas très utilisable dans la pratique: on veut en général que le processeur fasse autre chose en parallèle. On verra plusieurs solutions pour résoudre ce problème.

## 2.7. Feuille de codage

La feuille de codage CALM du PIC16F870 en www.didel.com/picg/picg87x/pics/Pic87xCalm.pdf contient toute l'information nécessaire pour se rafraîchir la mémoire quand on programme. La documentation du fabricant n'aide pas vraiment pour bien comprendre les instructions, mais elle est nécessaire pour les fonctions spéciales d'entrée-sortie.

JDN 30 novembre 1903