

Apprendre à programmer avec le 16F877A

--plus mis à jour – voir <http://www.didel.com/pic/Cours877-1.pdf> etc

2013 Nos nouveaux outils basés sur Arduino

www.didel.com/DiduinoPub.pdf

Le but est d'apprendre à écrire des programmes en assembleur pour des application utilisant un code efficace sur des microcontrôleurs en général plus petits. Notre démarche est de tester les programmes sur un 16F877 qui a 4 ports d'entrée-sortie, ce qui facilite la compréhension du fonctionnement. Le Microdual 16F877 est aussi idéal pour mettre au point des applications utilisant des processeurs en boîtier 6, 8, 14 pattes.

En plus de la carte 877, 4 cartes Microduals et des composants aident pour les tests.

La grande idée des Microduals est que vous ne les utilisez pas dans votre application. Ils restent toujours prêts pour tester des routines et préparer des nouveaux projets.

Notons encore que pour éviter d'allonger ce document avec des informations essentielles pour maîtriser tous les aspects d'un PIC, mais pas indispensables quand on n'a pas encore le besoin pour son application, beaucoup de renvois sont fait aux documents écrits depuis plus de 10 ans, qui se réfèrent parfois à des cartes de développement périmées ou des processeurs qui ne présentent plus d'intérêt. En particulier, le cours de 2002, partiellement repris ici, est souvent cité pour des points plus spécialisés. Voir www.didel.com/picq/picq87x/CoursPicq87x.html .

Une approche moins systématique avec des programmes pour clignoter et faire de la musique se trouvent sous www.bricobot.ch/programmer/CalmBimo.pdf

Matériel nécessaire

Pickit2 65.-

Kit Microdual A2840 complété 45.-

16F877A 10.-

Doc sous www.didel.com/pic/Prog877kit.pdf

La doc générale sur les Microduals est sous

www.didel.com/pic/Microduals.pdf

Nouveau : composants supplémentaires pour tester les programmes

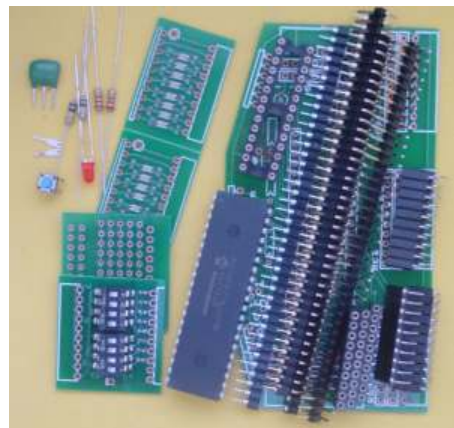
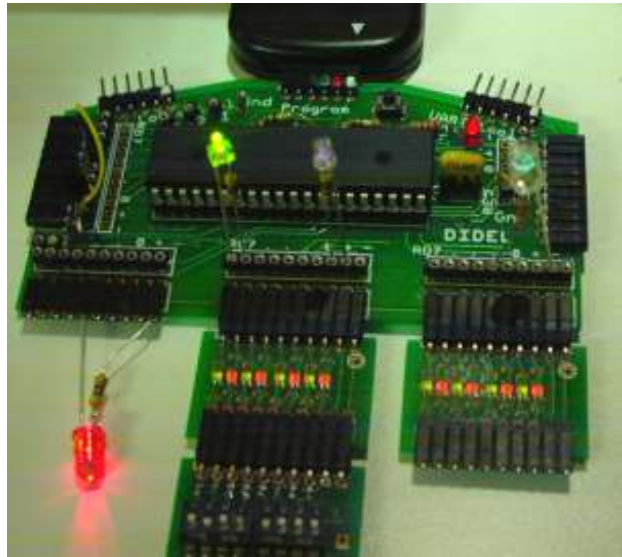


Table des matières

- 1 Introduction
- 2 Ports d'entrée-sortie
- 3 Toutes les instructions
- 4 Timers et interruption
- 5 Entrées analogiques
- 6 Commande de moteurs
- 7 Transferts série
- 8 EeProm et bootloader
- 9 Tables
- 10 Accès indirect
- 11 Séquencement et multitâche
- 12 Arithmétique
- 13 Macros et .lf
- 14 Structure et mise en page
- 15 Dépannage et compléments
- 16 Programmes de test
- 17 Routines et programmes exemple



1 Introduction

Avant de pouvoir faire les exercices de ce document, il faut installer

SmileNG www.didel.com/pic/SmileNG.zip

Pour la documentation de l'éditeur SmileNG et ses ordres de mise en page, voir

<http://www.didel.com/pic/SmileNG.pdf> .

Pickit2 www.bricobot.ch/docs/Microchip.zip

La procédure d'installation se trouve en www.didel.com/pic/InstallPic.pdf

A peu près la même chose, avec le brochage du connecteur de programmation sous

www.didel.com/pic/Pickit2.pdf

Si vous avez des problèmes avec un circuit qui refuse de se programmer, regardez

www.didel.com/pic/RecupPic.pdf

Les **exemples de programmes** associés à ce document se trouvent sous

www.didel.com/pic/Prog877Sources.zip Mettre ces fichiers dans un dossier Prog877.

Il faut ensuite connecter le Microdual 877, tester les sorties avec le programme T877Cli.hex (ce programme sera expliqué plus loin). Déplacer l'afficheur sur les ports pour vérifier que cela clignote partout.

1.1 Comment travailler ?

Chacun a besoin d'explications différentes pour bien comprendre. Ce document va trop lentement ou trop vite pour vous. Où trouver d'autres explications ? Le danger est de se disperser à lire des documents qui disent la même chose différemment et ne font pas progresser.

Si vous avez une incertitude concernant une instruction, le fichier

www.didel.com/pic/Calmlnstr877.pdf résume les instructions et l'effet sur les flags (section 3).

C'est bien de l'avoir toujours sous les yeux. Le Help de Smile est plus lent et un peu plus détaillé.

1.2 Notions de base : bits, octets, constantes, variables, registres, ...

Si ces notions sont nouvelles pour vous, le mieux est de lire la brochure Dauphin que vous pouvez obtenir gratuitement chez Zigobot et que vous pouvez lire sous

<http://www.epsitec.ch/Dauphin.zip>

Cette brochure vous apprend à programmer un processeur inventé, à la fois plus simple et plus riche que les PICs.

Le notions de nombre, constantes, variables, etc sont expliquées avec plus de détails dans le fichier www.didel.com/pic/Bases.pdf dont la lecture est recommandée maintenant ou plus tard, quand vous aurez fait quelques exercices.

1.3 Architecture des PICs

Les PICs ont une architecture simple, avec une mémoire d'instructions 12 ou 14 bits, des variables 8 bits et des périphériques 8 bits dans le même *espace* que les variables. Cet espace des variables est coupé en 4 banques, ce qui complique la sélection : on travaille principalement en banque 0 en passant brièvement dans d'autres banques lorsque c'est nécessaire. Comprendre l'architecture interne n'est pas essentiel. Si cela vous intéresse, la doc de Microchip est naturellement la référence. Le document www.didel.com/pic/Architecture.pdf présente les éléments importants, pour maintenant ou plus tard. Les périphériques ont plusieurs fonctions et il faut initialiser la bonne selon l'application. On expliquera les plus utiles dans des exemples. Les programmes PicTests www.didel.com/pic/PicTests.pdf évitent de se perdre dans la doc détaillée de Microchip pour les utilisations standard. Si vous voulez en savoir plus sur des bits qui définissent le comportement des périphériques, il faut chercher la documentation du fabricant sous ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf ou utiliser une traduction en français, parfois un peu améliorée, par [Bigonoff](#) ou [Oumnad](#). Les livres de [Tavernier](#) ont la même approche.

1.4 L'assembleur CALM

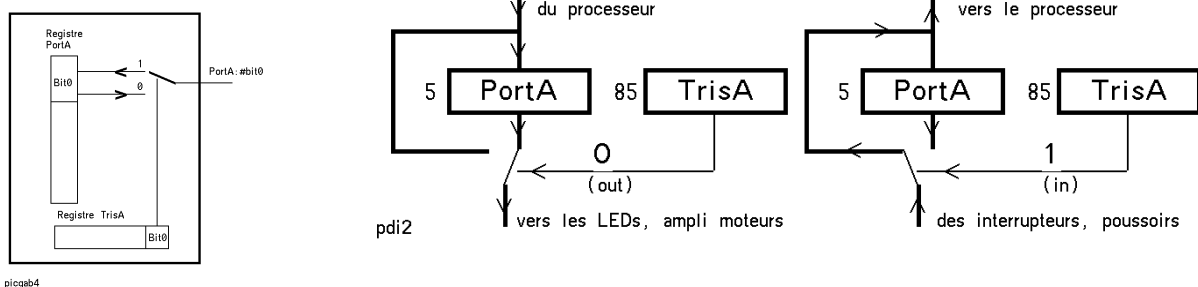
Les règles de l'assembleur et les pseudo-instructions apparaîtront petit à petit, parfois sans explications. Le Help de SmileNG vous donnera quelques éléments d'explication. Si une présentation systématique vous est nécessaire avant de voir des exemples de programme, lisez www.didel.com/pic/Calm.pdf

2. Entrées et sorties

L'intérêt des microcontrôleurs est qu'il ont des sorties sur lesquelles on peut brancher des diodes lumineuses, dans des amplis pour commander des moteurs, un haut-parleur. Ils ont des entrées pour connecter des poussoirs et interrupteurs, pour lire des signaux analogiques, pour mesurer des durées d'impulsions. Ces fonctions sont programmables et varient un peu d'un PIC à l'autre, influençant le choix du microcontrôleur et le câblage.

2.1 Registre de direction

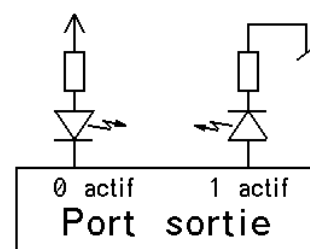
Pour décider si une broche du processeur est une entrée ou sortie, il faut que le contrôleur aie des aiguillages commandés par un registre de direction appelé Tris. Si le bit Tris est à 1 (input) la broche du circuit est connectée au bus et amène l'information dans le registre W, si l'instruction Move Port,W est exécutée. Si le bit Tris vaut 0 (output), c'est l'information qui vient du bus qui est mémorisée dans le registre et activée en sortie.



L'adresse du registre Tris est la même que pour le port correspondant, mais dans la banque 1, sélectionnée en activant le bit RP0 du registre Status. La section 2.6 montre comment programmer la direction.

2.2 Affichage de test LB8 et module d'entrée Sw8

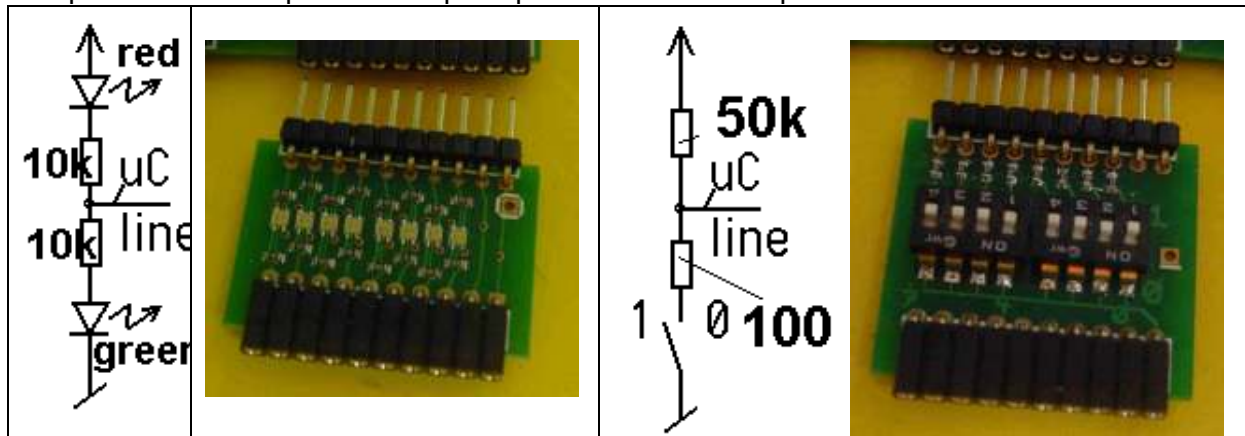
Pour visualiser l'état 0 ou 1 d'une sortie, une diode lumineuse est utilisée. Elle peut être branchée de deux façons différentes. Les transistors de sortie sont un peu meilleurs pour soutirer du courant (max 10 mA), donc on voit souvent dans des applications la solution avec la cathode vers le processeur, donc un zéro allume. La valeur de la résistance fixe la luminosité de la diode. Avec l'anode côté processeur, l'état 1 allume la diode..



Le micromodule d'affichage **Lb8** utilise des diodes bicolores et une paire de résistances qui affichent le zéro (en rouge), le 1 (en vert) et l'état "flottant" en entrée avec les deux diodes allumées en demi-intensité (on voit surtout le rouge).

Pour agir sur les entrées, on branche un interrupteur ou poussoir avec une résistance "pull-up".

L'interrupteur fermé relie à la masse, donc impose un zéro logique. On peut câbler l'inverse, avec une "pull-down" et imposer un "0" quand l'interrupteur est ouvert. Cela ne se fait pas car les processeurs ont parfois des "pull-up" internes et on en profite.



La correspondance entre position haut/bas de l'interrupteur et état logique 0/1 sur l'entrée du port doit être bien documentée. Pour le processeur, décider qu'un 0 est actif ou un 1 est actif, est tout aussi facile.

Le micromodule **Sw8** a 8 interrupteurs avec des pull-up de 1 kOhm et des résistances en série de 220 Ohm pour limiter le courant de court-circuit si l'interrupteur est fermé et la broche du contrôleur initialisée en sortie et à l'état "1". Sans résistance, le courant de court-circuit serait de 20mA environ, ce qui n'est heureusement pas destructeur si de courte durée.

2.3 Initialisation

L'initialisation de la direction des entrées-sorties dépend du schéma de l'application et se fait une fois au début. Une bonne habitude est de terminer cette initialisation par 3 clignotements. S'ils ont lieu, le programme a démarré correctement ; s'il ne continue pas juste, c'est une erreur dans votre programme.

Une autre initialisation, appelée configuration, est nécessaire pour dire par exemple si l'oscillateur est interne ou externe. Le mot de configuration doit être écrit à l'adresse 2007 et n'as pas de raison d'être changé dans nos applications, donc on ne va donc pas l'expliquer.

2.4 Exemple : osciller le portC

Le portA n'a que 6 bits dont un spécial (bit4). Le portB a deux lignes pour la programmation qui ont quelques contraintes. Sur les portC et D du 16F877 on peut faire ce qu'on veut et on va de préférence les utiliser.

Activer-désactiver le portC, donc créer une impulsion positive, s'écrit de plusieurs façons. Il faut naturellement avoir d'abord initialisé le portC en sortie, donc mis des "0" dans le registre TrisC, ce qui se fait tout au début du programme.

La première et la 3^e solutions sont identiques.

Move #2'11111111,W	Move #2'11111111,W	Move #-1,W
Move W,PortC	Move W,PortC	Move W,PortC
Move #2'00000000,W	Clr PortC	Move #0,W
Move W,PortC		Move W,PortC

On peut encore donner le paramètre en hexadécimal

2'11111111=16'FF 2'10100101 = 16'A5

Le décimal est utilisé pour initialiser des compteurs, mais il n'est pas logique pour repérer les bits d'un port. Toutefois, -1 = 16'FF = 2'11111111 est très pratique et jusqu'à 7, la conversion est immédiate : 6 = 2'00000110 = 16'06

Pour plus de détails sur le binaire et l'hexadécimal, voir www.didel.com/pic/Bases.pdf

Pour répéter ces instructions d'activation/désactivation, on crée une boucle en mettant une étiquette au début de la boucle et en ajoutant une instruction qui saute à cette étiquette, c'est à dire à l'adresse correspondante dans la mémoire du processeur.

```

Bcle:   Move    #2'11111111,W      ; 1 µs
        Move    W,PortC
        Move    #2'00000000,W
        Move    W,PortC
        Jump   Bcle                ; 2 µs
    
```

Les PICs à 4 MHz exécutent une instruction par microseconde (2 µs pour les sauts). Les 4 instructions précédentes vont activer le port C pour 2 microsecondes seulement, et désactiver pour 3 microsecondes. On ne verra rien sans utiliser un oscilloscope.

2.5 Boucles d'attente

Pour que le processeur modifie les sorties à une vitesse « humaine », il faut insérer des boucles d'attente. On initialise un compteur et on décompte jusqu'à la valeur zéro, que le processeur sait bien reconnaître.

```

Move #250,W
Move W,Cx1
Att: Dec Cx1 ; 1 µs
Skip,EQ ; 1 µs
Jump Att ; 2 µs

```

Ce bout de programme va faire 250 fois la boucle qui dure 4 microsecondes, donc la durée totale est de 1 milliseconde (plus les 2 microsecondes d'initialisation du décompteur).

A noter les trois nouvelles instructions. Dec Cx1 décompte la variable mentionnée. Comme le compteur d'une voiture qui roulerait en marche arrière, le compteur ne se bloque pas en zéro, mais continue avec la valeur maximum. L'instruction Skip,EQ permet de savoir si le compteur est à zéro (EQ equal). Si oui, le processeur saute l'instruction suivante. Si non, il exécute le saut qui lui fait attendre encore un tour.

C'est encore trop rapide pour notre oeil. On ne peut pas augmenter la valeur 250 au delà de 255, car la variable Cx1 est un mot de 8 bits et les PICs n'ont pas d'instructions pour des compteurs ou décompteurs plus grands que 8 bits.

On fait alors une 2^e boucle qui appelle la première et utilise une 2^e variable. Vous voulez un dixième de secondes ? Il suffit de répéter 100 fois la boucle de 1ms.

```

; Boucle 100 millisecondes
Move #100,W
Move W,Cx2
Att2 :
Move #250,W
Move W,Cx1
Att: Dec Cx1 ; Boucle 1ms
Skip,EQ
Jump Att
Dec Cx2
Skip,EQ
Jump Att2

```

On pourrait naturellement compter et comparer quand on atteint la valeur 100 ou 250. C'est moins efficace avec le PIC.

2.6 Programme complet

Il faut donc insérer nos boucles d'attente de 100ms après avoir écrit et après avoir effacé le motif sur le PortC. Le programme doit aussi initialiser le PortC en sortie, et déclarer l'adresse des variables Cx1 et Cx2. Il faut enfin configurer le processeur.

Charger le programme T877Cli0.asm dans SmileNG et assembler (F5). L'assembleur convertit ce programme en binaire, et crée un fichier avec l'extension .hex. Il faut le transférer dans le Pickit2 et on peut voir le code dans la fenêtre "Program memory". Exécuter en cliquant sur On à la fin du "Write".

<pre> \prog;T877Cli0 Clignote les sorties du portC \b; RC0..RC7 oscille, ;période 0.65 s a 4 Mhz ; Ne pas utiliser si des sorties sont court-circuitées ou connectées a des sorties d'interface ; Voir T877Cli pour une écriture plus élégante .Proc 16F877 .Ref 16F877 CX1 = 16'20; Début des var CX2 = 16'21 Motif1 = 16'55; 2'01010101 Motif2 = 16'AA; 2'10101010 </pre>	<pre> .Loc 0 Deb: Move #0,W ; out Move W,TrisC Loop: Move #Motif1,W Move W,PortC ; attente 100ms Move #100,W Move W,Cx2 Att2: Move #250,W Move W,Cx1 Att: Dec Cx1 Skip,EQ Jump Att Dec Cx2 Skip,EQ </pre>	<pre> Move #Motif2,W Move W,PortC ; attente 100ms Move #100,W Move W,Cx2 Att2b: Move #250,W Move W,Cx1 Attb: Dec Cx1 Skip,EQ Jump Attb Dec Cx2 Skip,EQ Jump Att2b Jump Loop ;Configuration .Loc 16'2007 .16 16'3F39 </pre>
---	--	---

2.7 Optimisation ou simplifications

L'instruction DecSkip,EQ remplace les deux instructions de la boucle d'attente.

Si on n'a pas besoin d'une durée précise, on peut ne pas recharger les compteurs. Après être arrivés à zéro, ils recommencent un tour de 256 pas. La façon la plus simple de faire une attente de $256 \times 256 \times 3 \mu s \approx 0.2$ secondes est d'écrire

```
A$: DecSkip,EQ Cx1
    Jump A$           ; A$ est une étiquette dite locale
    DecSkip,EQ Cx2
    Jump A$
```

Le programme **T877Cli.asm** utilise cette nouvelle attente et clignote tous les ports en inversant le motif avec un ou-exclusif que l'on comprendra plus loin. Le programme est nettement plus court. On remarque que l'initialisation du portA nécessite d'agir sur le registre AdCon1. On reparlera de l'anomalie des registres Tris et de la 2^e banque.

T877Cli.asm est un programme de test que l'on recharge toutes les fois que l'on veut vérifier que le processeur fonctionne et que toutes les sorties sont connectées. S'il ne tourne pas, c'est que l'oscillateur interne n'est pas initialisé.

2.8 Routines

Ce n'est pas très élégant, ni très efficace, de devoir écrire deux fois la même boucle d'attente, comme dans le programme **T877Cli0.asm**. On écrit une seule fois le module, appelé routine, et on peut l'appeler autant de fois que l'on veut. L'instruction Call fait l'appel et à la fin de la routine, l'instruction Ret (return) retourne à l'instruction qui suit le Call.

Ecrivons une routine **AttWx1ms** avec un paramètre en entrée.

```
\rout:AttWx1ms|Attente de (W) fois 1ms
\in:W nombre de ms
\mod:W Cx1 Cx2
AttWx1ms: Move    W,Cx2
A$:  Move  #250,W
     Move  W,CX1
B$:  Nop
     DecSkip,EQ CX1   ; boucle 1ms
     Jump  B$
     DecSkip,EQ Cx2   ; boucle (W) fois 1ms
     Jump  A$
     Ret
```

Cette routine est le premier élément d'une bibliothèque de routines que chaque programmeur se constitue. Les 3 premières lignes contiennent toute l'information nécessaire pour l'utiliser.

2.9 Agir sur une seule ligne de sortie

On veut souvent modifier une sortie sur un port sans modifier celles qui sont à côté sur le même port. Une première solution est de lire le port, modifier avec les bonnes instructions, et ré-écrire la valeur. Par exemple, si on veut mettre à 1 le signal LedV du portC (sur le bit2 en 3^e position) on peut écrire

```
Move  PortC,W
Or    #2'00000100,W   ; force le bit2 à 1, ne modifie pas les autres
Move  W,PortC
```

Une instruction fait exactement cela :

```
bLedV = 2 ; 3e position car on numérote 0 1 2 .. depuis la droite
Set   PortC:#bLedV ;
```

Pour mettre le bit bLedV à zéro (ce qui allumera ou éteindra la LED verte selon le câblage, il faut écrire :

```
Move  PortC,W
And   #2'11111011,W ; force le bit2 à 0, ne modifie pas les autres
Move  W,PortC
```

Une instruction fait exactement cela :

```
Clr   PortC:#bLedV
```

Si le ET (And) et le OU (Or) logique ne vous sont pas familiers, regardez le document www.didel.com/pic/Bases.pdf. Le Ou exclusif du programme **T877Cli.asm** y est aussi expliqué.

2.11 Lire un port

Pour mettre un port en entrée, il suffit de mettre des 1 partout dans le registre Tris associé.

```
Set   Status:#RP0
Move  #2'11111111,W
Move  W,PortC      ; TrisC
Clr   Status:#RP0
```

Copier un port sur un autre s'écrit simplement

```
Move  PortC,W      ; initialisé en entrées
Move  W,PortD      ; initialisé en sorties
```

Il n'y a pas besoin de boucle d'attente dans ce cas, mais il peut y avoir de bonnes raisons d'en mettre une.

Le programme **T877CopyCD.asm** lit les interrupteurs sur le portC et copie sur le portD.

Modifier ce programme pour lire sur le PortB. Il peut y avoir problème, car le programmeur utilise RB6 et RB7 pour écrire dans le 16F877. Un interrupteur fermé peut perturber, malgré la résistance série.



2.12 Lire un poussoir isolé

On peut naturellement lire tout le port et ensuite isoler le bit pour savoir s'il vaut un ou zéro.

L'instruction AND permet ce masquage d'un ou plusieurs bits dans un mot.

Par exemple, un poussoir est connecté sur le bit4 du port C initialisé en entrée. Le poussoir actif ferme le contact et impose un zéro sur l'entrée.

```
bPous = 4
Move  PortC,W
And   #2'00010000,W      ; seul le bit 4 reste dans son état
Skip, EQ      ; le résultat est nul si le bit 4 est à zéro
Jump  PousInactif
Jump  PousActif
```

On doit très souvent savoir si un bit d'un port ou d'une variable est à un ou zéro, une instruction unique remplace les 3 instructions de masquage d'une ligne:

```
TestSkip,BC PortC:#bPous  (skip if bit clear) saute si bit 4=0
Jump  PousInactif
Jump  PousActif
```

Pour tester si un bit est à un, on a l'instruction

```
TestSkip,BS PortC:#bPous  (skip if bit set)
Jump  PousActif
Jump  PousInactif
```

Les trois instructions (Move – And – Move) sont toujours utilisées si on veut tester un groupe de bits. Pare exemple, si 4 poussoirs sont pressés, le AND permet de savoir qu'il y en a un qui est activé, mais il faut ensuite décider lequel.

2.13 Mélange d'entrées et sorties sur un même port

Le registre Tris permet de mélanger des entrées et sorties, mais il peut y avoir des directions prioritaires, par exemple si le convertisseur analogique est déclaré. Les programmes de test mettront en évidence quelques cas simples. La documentation du fabricant sur 300 pages les décrit tous !

Comme exemple, câblons une LED sur le bit 2 du PortC et un poussoir sur le bit 4. Décidons que les 4 bits de poids faibles sont des sorties, et les 4 bits de poids forts sont des entrées.

Les instructions spécifiques à cet exemple sont :

```
bLed  = 2
bPous = 4
DirC  = 2'11110000
...
Move  #DirC,W
Move  W,TrisC      ; en banque 1
...
TestSkip,BS PortC: bPous
Clr   PortC :#bLed
```

```
TestSkip,BC PortC: bPous
Set PortC :#bLed
```

Le programme **T877SwapC.asm** est plus intéressant, car la moitié gauche du portC est en entrée et la moitié droite en sortie. L'instruction

```
Swap PortC,W
```

permute les 2 moitiés en transférant dans W.

Question : peut on remplacer les 2 instructions

```
Swap PortC,W
Move W,PortC
```

par

```
Swap PortC
```

Essayez, ça marche en fait pour toutes les variables ; c'est parfois pratique de pouvoir permuter les deux moitiés..



Notons que le portA a des entrées analogiques, en service par défaut à l'enclenchement. Il faut désactiver (positionner un aiguillage pour mettre le convertisseur hors circuit) avec les 4 instructions que l'on voit dans le programme **T877Cli.asm**:

```
Clr PortA
Set Status:#RP0
Move #16'06,W
Move W,AdCon1
```

On a passé en banque 1, il ne faudra pas oublier de revenir!

2.14 Pull-ups

Sur une entrée non connectée, le potentiel n'est pas bien défini. Si on connecte un poussoir, il faut une résistance qui impose une tension sur l'entrée, en général le +5V (état 1), et le poussoir court-circuite avec la masse (état 0).

Cette résistance peut être programmée de façon plus ou moins élégante selon le processeur. Sur les anciens processeurs comme le 16F877, un seul bit met en service les pull-ups du port B seulement. Sur les processeurs plus récents, des registres permettent de commander individuellement toutes les sorties avec ou sans pull-ups.

Comment vérifier ? Initialisons le portB en entrée, avec copie sur le portC.

Ajoutons deux instructions qui connectent des pull-ups sur les entrées du portB.

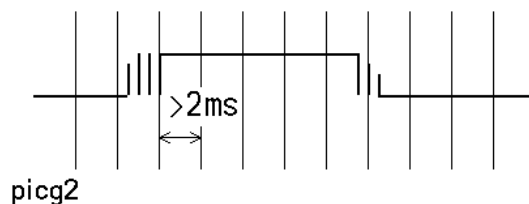
```
Move #0,W
Move W,Option ; ce registre est spécial, on en reparlera
```

On remarque l'effet en touchant les lignes du portB avec les doigts, l'affichage sur le portC reste à 1. Sans les pull-ups, (et sans interrupteurs ou affichage microdule) le doigt amène du "220V" qui fait varier les entrées, donc la copie sur le portC.

2.15 Rebonds de contact

Si on veut compter les actions d'un poussoir, le programme ne doit compter qu'une fois par action, et ne pas compter quand on presse, ni quand on relâche. Il faut donc deux boucles d'attente. L'interrupteur Microdule n'est pas pratique comme poussoir, on peut en câbler un avec 3 fils (+ pour la pull-up, -, signal).

Les contacts mécaniques ont des rebonds, pendant 0.1 à 2 millisecondes et le processeur peut les voir comme des actions séparées si on échantillonne trop souvent. On échantillonne donc à plus de 2ms, et moins de 0.1s pour ne pas rater les actions humaines les plus rapides.



Le cœur du programme pour compter contient donc des boucles d'attente de 20ms (routine Del20ms) et s'écrit

```
AtPous : Call Del20ms
TestSkip,BS PortC :#bPous
Jump AtPous ; On attend l'action
Inc Compte
```



```

AtRel : Call    Del20ms
        TestSkip,BC PortC :#bPous
        Jump    AtRel      ; On attend le relâchement
        Jump    AtPous     ; On recommence

```

Pour visualiser le compteur, on le copie par exemple dans le portD initialisé en sortie. On peut aussi directement utiliser le portD comme compteur et remplacer Inc Compteur par Inc PortD.

Le programme **T877CntPous.asm** permet de tester ces rebonds.
Mettez un ; (commentaire) devant les instructions Call Del20ms. Le processeur va compter les rebonds de plus de quelques microsecondes.
Ajoutez dans le logiciel le test d'un bit du portC pour faire une remise à zéro du compteur.
Vous ne voyez pas comment ? Dans la boucle AtPous, testez un autre bit et s'il est actif, il faut exécuter l'instruction Clr PortD.



2.16 Tester la durée d'action sur un poussoir

On veut savoir combien de temps, ou quand on a pressé sur un poussoir. Il faut dans la boucle d'attente, surveiller le signal du poussoir, toutes les 50 millisecondes au moins si on veut être assez précis, et pas trop souvent si on ne veut pas mesurer la durée d'un rebond de contact. Si on veut mesurer une durée, pour voir par exemple comment on peut être rapide, on met un compteur à zéro, on attend quelques secondes avant d'allumer une LED et on compte à partir de cet instant. Le cœur du programme est donc :

```

TestReflexe:
    Clr    PortC:#bLed
    Call   Attente      ; attente 1-5 secondes
    Set    PortC:#bLed
    Clr    PortD
AtPous:    Call Del20ms
            Inc    PortD ; Durée en multiple de 20ms
            TestSkip,BS PortC :#bPous
            Jump   AtPous
Fini:     Jump   Fini ; Jump APC idem, car APC = adresse de l'instruction

```

On se pose naturellement beaucoup de questions avec ce programme. Comment compter en décimal et pas en binaire, comment mettre un affichage 7 segments, comment faire un programme attractif. Cela viendra !

On voit à la fin de ce programme l'instruction Jump Fini ; il n'y a plus rien à faire pour nous, mais si on ne met pas cette instruction, le processeur va prendre les positions mémoire suivantes comme des instructions, et on ne peut pas deviner ce qui va se passer.

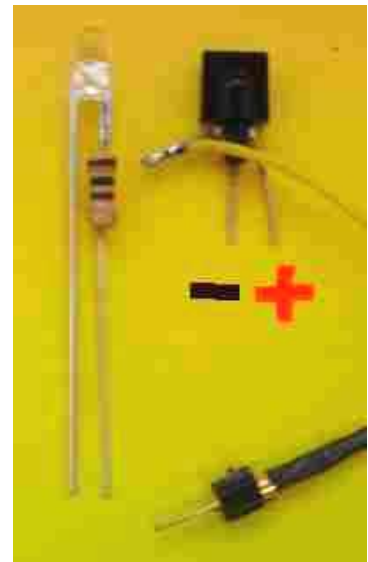
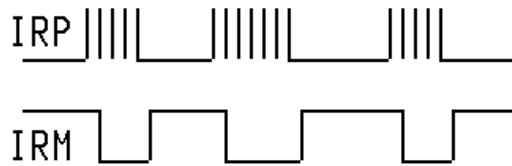
2.17 Musique

Jouer une fréquence audible se fait avec une boucle d'attente.
N'importe quel haut-parleur avec une résistance en série de 100 Ohm peut être branché entre la sortie d'un port et le +5V.
Un 0 va attirer la membrane, un 1 la relâcher. A basse fréquence (avec le programme T877Cli) on entend un clic lorsque la membrane est attirée et relâchée. A fréquence audible, l'intensité est parfois très variable à cause des fréquences de résonance du haut-parleur.
Plusieurs programmes sont expliqués sous www.didel.com/bot/sons/Sons.doc et les programmes cités peuvent être trouvés sur le site et facilement adaptés (il faut supprimer le .Ins Dev877.asi , remplacer le IOFF par l'initialisation du port sur lequel est câblé le HP, et remplacer (ou créer) les macros HpOn HpOff.



2.17 Infrarouge

Les télécommandes infrarouges transmettent des trains d'impulsion à 36-40 kHz. Un circuit IRM (Infra Red Module) filtre et donne sur sa sortie l'enveloppe du signal, en impulsions négatives de 0.8 à 3 ms en général.



Le plus simple est de brancher l'IRM sur une entrée, et de reconnaître l'action d'une télécommande de TV quelconque, comme si c'était un poussoir avec des rebonds. Essayez le programme **T877CntPous.asm** avec une attente à définir. Un oscilloscope est nécessaire pour mettre au point des programmes non triviaux. Le principe des télécommandes Emir est expliqué brièvement sous www.didel.com/lr/EmirSpecs.pdf

Les télécommandes PicoZ sont expliquées sous <http://www.didel.com/lr/lrControl.pdf>.

Le format RC5 utilisé dans la plupart des télécommandes commerciales est abondamment documenté sur le Web.

3. Toutes les instructions

Rappelons que chaque instruction a éventuellement un effet sur des bits stockés dans le registre d'état (status). Le bit Z (ou flag Z) indique si l'octet transféré est nul (zéro). Il n'est pas activé par toutes les instructions qui devraient ou pourraient le faire, il faut donc vérifier sur la feuille de codage avant d'imaginer soi-même la logique du processeur.

Le bit C indique qu'une opération arithmétique a eu un dépassement de capacité. Le PIC ne réagit pas comme d'autres processeurs et il faudra faire attention avant de tester ce bit.

Le bit D est exceptionnellement utilisé pour les calculs en décimal.

La feuille de codage compacte est en dernière page. www.didel.com/pic/CalmInstr877.pdf

Imprimez quelques copies, c'est indispensable d'en avoir toujours une sous la main pour vérifier que l'instruction existe et quel est son effet sur les flags. Sur cette feuille et ci-dessous, les flags modifiés sont indiqués entre crochets. En première colonne, la notation de microchip est rappelée pour ceux qui y sont familiarisés. Le passage de Microchip à CALM est facile, mais il y a des variantes de l'assembleur Microchip avec différentes notations.

3.1 Instructions de déplacement (Move)

On notera que le flag Z n'est pas toujours modifié.

MOVLW VAL	Move #Val,W	[none]
MOVWF REG	Move W,Reg	[none]
MOVF REG,0	Move Reg,W	[Z]
MOVF REG,1	Test Reg	[Z]

Rappelons que le signe dièse # précède une valeur numérique. S'il est oublié, le processeur va chercher une variable à cette position. L'assembleur signalera une erreur si le nombre est supérieur à 16'7F, puisque le processeur ne peut pas adresser plus de 127 variables. Autrement le programme a de bonnes chances de dérailler.

Des instructions spéciales agissent sur le registre OPTION et sur les registres de direction des PICs à 28 broches au plus.

TRIS PORTA	Move W,TrisA	[none]
TRIS PORTB	Move W,TrisB	[none]
TRIS PORTC	Move W,TrisC	[none]
OPTION	Move W,Option	[none]
CLRWDT	ClrWDT	[none]
SLEEP	Sleep	[none]

Pour les instructions très spéciales ClrWdt et Sleep, consultez la documentation de Microchip.

Note : les trois instructions **Move W,TrisA** etc seront remplacées prochainement par **TrisA** etc et TrisA B C D E deviendront des adresses en banque1 avec les mêmes valeur numériques que PortA B C D E. Attention, actuellement **Move W,TrisD** **Move W,TrisE** sont

acceptés par l'assembleur mais n'ont pas d'effet. Ces instructions seront correctes quand l'assembleur sera modifié, mais en banque 1 seulement.

3.2 Instructions logiques (AND OR XOR)

Toutes ces instructions modifient le bit Z ($Z = 1$ si le résultat est zéro).

```

ANDLW  VAL          And  #Val,W      [Z]
ANDWF  REG,0        And  Reg,W       [Z]
ANDWF  REG,1        And  W,Reg       [Z]
IORLW  VAL          Or   #Val,W      [Z]
IORWF  REG,0        Or   Reg,W       [Z]
IORWF  REG,1        Or   W,Reg       [Z]
XORLW  VAL          Xor  #Val,W      [Z]
XORWF  REG,0        Xor  Reg,W       [Z]
XORWF  REG,1        Xor  W,Reg       [Z]

```

Rappelons que le ET logique (And) garde les bits qui sont à un dans les deux opérandes.

Le OU (Or) garde tous les bits.

Le Ou exclusif (XOR) garde les bits qui sont complémentaires (différents).

On utilise souvent le XOR pour décider si les deux nombres/contenus de registres sont égaux.

```

11010010      11010010      11010010      11010010
10101010      10101010      10101010      11010010
-----
And 10000000      Or 11111010      Xor 01111000      Xor 00000000 (Z=1)

```

3.3 Instructions arithmétiques (Add,Sub)

L'addition modifie les trois bits d'état [C,D,Z].

```

ADDLW  VAL          Add  #Val,W      [C,D,Z]  Val + (W) → W
ADDWF  REG,0        Add  Reg,W       [C,D,Z]  (Reg) + (W) → W
ADDWF  REG,1        Add  W,Reg       [C,D,Z]  (W) + (Reg) → Reg

```

La notation (W) signifie contenu de W. Le processeur additionne les deux valeurs, transférées dans le registre W.

```

      +      +
      11010010
Add   10101010
-----
[C=1] 01111100

      +
      11000010
Add   00101010
-----
[C=0] 11101100

+++++++ reports
      01010010
Add   10101110
-----
[C=1 z=1) 00000000

```

Pour la soustraction, CALM utilise la notation Motorola: le 1^{er} opérande est soustrait du second. Contrairement à la majorité des processeurs, les PICs font la soustraction par addition du complément à 2.

```

SUBLW  VAL          Sub  W,#Val,W    [C,D,Z]  Val + -(W) → W
SUBWF  REG,0        Sub  W,Reg,W     [C,D,Z]  (Reg) + -(W) → W
SUBWF  REG,1        Sub  W,Reg       [C,D,Z]  (Reg) + -(W) → Reg

```

(equiv. to Sub W,Reg,Reg)

```

      - -
      11010010
Sub   10101010
-----
[C=1] 00101000

      ++ + ++
      11010010  (#Val ou Reg)
Add   01010110  + (-W)
-----
[C=1] 00101000

```

On voit que le Carry est inversé par rapport à l'opération usuelle de soustraction.

La soustraction d'une valeur immédiate est très différente de tous les autres processeurs et la notation CALM montre bien que l'on prend la valeur et soustrait le contenu de W. Si l'on doit soustraire une valeur à une variable, on ajoute le complément à 2 de la valeur:

```
Add #-Val,W
```

La soustraction est souvent utilisée pour comparer deux variables (c'est à dire les nombres 8 bits contenus dans ces variables):

```
Move Var1,W
Sub W,Var2,W
```

Si le carry est à un (CS Carry Set), Var2 est supérieur à Var1. Si le bit Z est à un, les deux nombres sont égaux.

Les soustractions, comparaisons et gestion des nombres négatifs peuvent être étudiées quand on en a besoin, voir en particulier www.didel.com/pic/Arith.pdf. C'est assez rare ; un microcontrôleur manipule des bits, mesure des temps, passe par des tables, et n'est pas fait pour calculer !

3.4 Incrémentation/décrémentation (INC, DEC)

INCF REG,1	Inc	Reg	[Z]	(Reg)+1 → Reg
INCF REG,0	Inc	Reg,W	[Z]	(Reg)+1 → W, (Reg) → Reg
DECF REG,1	Dec	Reg	[Z]	(Reg)-1 → Reg
DECF REG,0	Dec	Reg,W	[Z]	(Reg)-1 → W, (Reg) → Reg

Attention, **Inc Reg,W** ne modifie pas **Reg**. Si l'on veut à la fois incrémenter Reg et en avoir une copie dans W, il faut deux instructions, incrémenter et copier

3.5 Complément et effacement (NOT, CLR)

COMF REG,1	Not	Reg	[Z]	inv(Reg) → Reg
COMF REG,0	Not	Reg,W	[Z]	inv(Reg) → W, (Reg) → Reg
CLRF REG	Clr	Reg	[Z=1]	0 → Reg
CLRW	Clr	W	[Z=1]	0 → W

Le complément (complément à 1) est une inversion de tous les bits. On peut le faire avec un Ou exclusif, ce qui est nécessaire pour inverser le registre W.

```
Xor #16'FF,W      Xor #2'11111111,W      Xor #-1,W
```

Le complément arithmétique (complément à 2) est différent. C'est la différence à zéro que l'on obtient par soustraction.

```
Sub W,#0,W
```

On peut aussi appliquer la règle que le complément à 2 est égal au complément à 1 plus 1, ce qui convient mieux pour complémentariser une variable.

```
Not Reg
Inc Reg [Z] -(Reg) --> Reg
```

La notion de complément à 2 est liée aux nombres négatifs. Prudence !

3.6 Rotation et permutation (RRC, RLC, Swap)

Les décalages à droite ou à gauche se font à travers le carry.

RRF REG,1	RRC	Reg	[C]	
RRF REG,0	RRC	Reg,W	[C]	(Reg) not modified
RLF REG,1	RLC	Reg	[C]	
RLF REG,0	RLC	Reg,W	[C]	(Reg) not modified
SWAPF REG,1	Swap	Reg		
SWAPF REG,0	Swap	Reg,W		(Reg) not modified

Attention de nouveau, **RRC Reg,W** ne modifie pas **Reg**.

Il faut en général préparer le carry avant de faire un décalage si on ne veut pas injecter une valeur aléatoire à droite ou à gauche.

Par exemple pour décaler à droite un nombre de 16 bits contenu dans deux variables, on écrit

```
ClrC
RRC High
RRC Low
```

Swap permute les deux moitiés d'un mot de 8 bits. Cela ne remplace pas 4 décalages, puisque la rotation se fait au travers du carry et nécessite 9 décalages pour un tour complet.

```
11010010 swap → 00101101
```

3.7 Instructions sur des bits

Le PIC est très performant pour forcer ou tester un bit sur un port ou une variable. L'instruction doit spécifier la variable, et le numéro du bit, qui est une valeur immédiate, donc précédée d'un signe #.

BCF REG,bNumber	Clr	Reg:#bNumber	[none]
BSF REG,bNumber	Set	Reg:#bNumber	[none]
BTFSC REG,bNumber	TestSkip,BC	Reg:#bNumber	[none]
BTFSS REG,bNumber	TestSkip,BS	Reg:#bNumber	[none]

Des exemples ont été vus précédemment.

L'instruction **Not Reg:#bNumber** qui permettrait d'inverser un bit manque souvent. On la remplace en faisant intervenir un OU exclusif, ce qui malheureusement modifie W:

```

Move #2**bNumber,W
Xor W,Reg

```

Les instructions logiques (And, Or, Xor) peuvent de façon similaire agir sur plusieurs bits à la fois pour mettre à un, mettre à zéro ou inverser.

Le registre de status qui contient les indicateurs C, D et Z peut être modifié par des set et clr bits. Calm documente les instructions suivantes, pratiques à utiliser:

```

BSF STATUS,0      SetC [C=1]   Set carry
BCF STATUS,0      ClrC [C=0]   Clr carry
BSF STATUS,1      SetD [D=1]   Set D, decimal carry flag
BCF STATUS,1      ClrD [D=0]   Clr D
BSF STATUS,2      SetZ [Z=1]   Set Zero flag
BCF STATUS,2      ClrZ [Z=0]   Clr Zero flag
BTFSC STATUS,0    Skip,CC      Skip if Carry Clear
BTFSS STATUS,0    Skip,CS      Skip if Carry Set
BTFSC STATUS,1    Skip,DC      Skip if Digit carry Clear
BTFSS STATUS,1    Skip,DS      Skip if Digit carry Set
BTFSS STATUS,2    Skip,EQ      Skip if Equal
BTFSC STATUS,2    Skip,NE      Skip if Non Equal

```

3.8 Instructions de saut

L'instruction de saut a une limitation d'adresse à 1k (10 bits d'adresse). Il y a naturellement un moyen pour étendre l'accès dans les PICs ayant plus de mémoire.

```

GOTO ADDR      Jump Addr

```

Il n'y a pas dans les PICs des instructions de saut conditionnelles comme dans les autres processeurs. Le Skip conditionnel est un peu moins agréable à écrire, mais tout aussi efficace.

```

INCF SZ REG,0    IncSkip,EQ Reg [none]
                  Increment Reg and Skip if result is Equal to zero
INCF SZ REG,1    IncSkip,EQ Reg,W [none] (Reg) not modified
                  Copy Reg in W, then increment W and Skip if
                  result is Equal to zero.
DECFSZ REG,0     DecSkip,EQ Reg [none]
DECFSZ REG,1     DecSkip,EQ Reg,W [none] (Reg) not modified

```

On remarque que les instructions IncSkip,NE DecSkip,NE n'existent pas. On les remplace facilement par deux instructions:

```

Inc Reg
Skip,NE

```

3.9 Appel de routines (Call, Ret, RetI)

On ne peut imbriquer que 8 appels de sous-programmes. C'est rarement une limitation, sauf sur les processeurs 12F et 10F qui n'ont que deux niveaux et où cela alourdit parfois l'écriture.

```

CALL Addr      Call Addr      (call routine)
RET            Ret              (return from subroutine)
RETFIE        RetI             (return from interrupt)
RETLW Val     RetMove #Val,W   (load W and return)

```

L'instruction RetI (return from Interrupt) active le bit GIE (General Interrupt enable) en plus de recharger l'adresse de retour (section 4).

L'instruction RetMove #Val,W est très intéressante, on le verra plus loin. Elle charge une valeur dans W au moment du retour, comme la syntaxe CALM l'exprime clairement.

3.10 Banques de registres et pages mémoire

L'architecture des PICs a été développée en 1973 quand la mémoire se mesurait en bytes et pas en gigabytes. Avec leurs instructions de 12 ou 14 bits, les PICs sont très efficaces pour des programmes courts avec peu de variables. Des trucs permettent d'augmenter ces capacités d'adressages limitées, mais s'il vous faut juste un peu plus de variables et de mémoire, apprenez à programmer efficacement, et s'il en faut beaucoup plus, changez de processeur !

Les PICs ont jusqu'à 4 banques de registres, de 128 octets au maximum, dans lesquelles se trouvent les périphériques et leurs registres de contrôle, plus des variables selon le processeur.

On change de banque en modifiant des bits dans le registre Status. Le plus clair est de définir des macros comme dans l'exemple I2C donné plus loin. Avoir des variables dans plusieurs banques alourdit le programme. Notre conseil est que le programme principal ne travaille qu'avec des variables en banque 0, les commutations de banques n'ayant lieu qu'à l'intérieur de routines bien définies.

Le compteur d'adresse des PICs n'a que 11 bits, limitant les sauts dans 2k de mémoire. Le document www.didel.com/picg/doc/DocPage.pdf montre comment passer à des adresses supérieures.

Il y a encore dans la mémoire des PICs des frontières toutes les 256 instructions que l'instruction géniale `add w, PCL` ne sait pas franchir.

Il y a enfin dans les PIC 10F et 12F avec instructions 12 bits des limitations à étudier dans les documents www.didel.com/picg/doc/DopiComp.pdf et www.didel.com/pic/Compati200.pdf

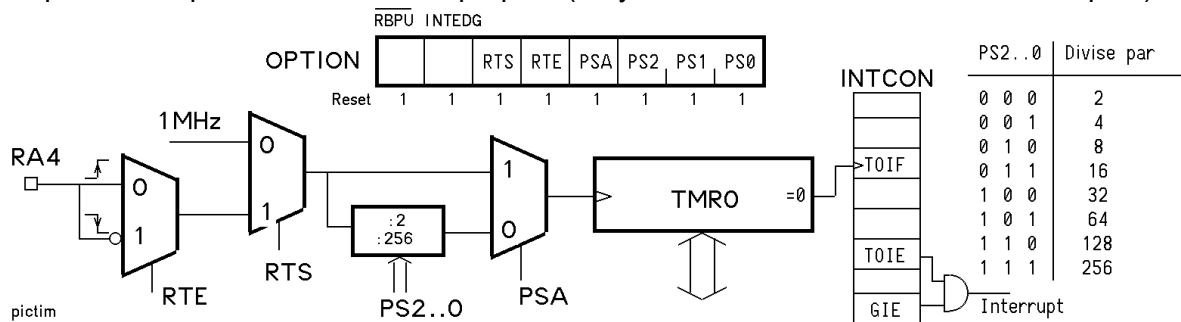
4. Timers et interruptions

Les PICs ont 1,2,3 timers. Ce sont des compteurs avec de la logique autour pour compter plus ou moins vite et mesurer ou générer des signaux. Ce document se limite au TMR0, qui existe dans tous les PICs 10F, 12F, 16F. Les autres timers doivent être étudiés avec les programmes de test mis à disposition et avec la documentation du fabricant ou des traductions.

4.1 Timer 0

Le timer 0 est un exemple simple d'un circuit programmable. Le TMR0 est un compteur que le programme peut lire ou écrire comme une variable. Il tourne toujours, et quand il déborde, il active une bascule, appelée TOIF (Timer Overflow Flag). Cette bascule est dans le registre IntCon.

Le compteur est précédé d'un prédiviseur programmable, c'est à dire que l'oscillateur qui fait tourner le processeur peut être divisé par une valeur codée sur 3 bits pour avoir un compteur qui tourne à 1 Mhz, 0.5 Mhz, ... 3.9 Khz (processeur à 4 Mhz). Il y a encore une possibilité pour compter des impulsions extérieures propres (s'il y a des rebonds, ils seront tous comptés).



Le schéma-bloc est très clair. On voit que pour compter le front descendant d'un signal câblé sur RA4 sans prédivision, il faut charger 2'00111000 dans Option.

Testons la fonctionnalité du TOIF en incrémentant un compteur (PortD) quand le TMR0 déborde. On compte si TOIF est actif, et on efface immédiatement TOIF (on ré-arme la bascule), donc on ne compte qu'une fois par tour.

```

Move #2'00000111,W      ; prédiviseur max
Move W,Option
Clr TMR0
Clr Option :#TOIF
Clr PortC
At$: TestSkip,BS Intcon:#TOIF
Jump Ato ; on attend
Clr Intcon:#TOIF
; on pourrait initialiser TMR0 à une valeur qui raccourcirait le cycle
Inc PortC
; on pourrait utiliser le PortC comme compteur 16 bits, (voir T877T0c.asm)

```

Jump At\$

Le programme **T877Tmr0.asm** permet de tester. Modifiez la valeur de prédivison dans Option.

4.2 Mesures de durées

Le timer peut compter plus vite qu'une boucle d'attente, et il compte sans ralentir le programme. Pour mesurer la durée d'une impulsion sur le poussoir (ou l'impulsion plus rapide d'une télécommande infrarouge), on met le compteur à zéro au début de l'impulsion et on attend la fin de l'impulsion pour faire la mesure.

```
; le signal est sur bPous sur le portC
Atp: TestSkip,BS PortC :#bPous
     Jump  Atp  ; on attend l'action
     Clr   TMR0 ; Prédiviseur selon les durées à mesurer
Atr: TestSkip,BC PortC :#bPous
     Jump  Atr
     Move  TMR0,W
     ; W contient la durée. On peut l'afficher ou l'utiliser.
```

A vous de faire le programme, cette fois ! Mais attention, le temps maximum mesurable est 256X255 microsecondes = 60ms, trop peu pour un poussoir. Il faut ajouter un compteur, qui mesure par exemple le centième. Le truc est expliqué plus loin si vous ne voyez pas..

4.3 Timer par interruption

Dans l'exemple précédent, on voit que le timer augmente la précision de la mesure, mais le programme est bloqué à surveiller le poussoir.

L'interruption permet d'avoir une tâche qui s'exécute quand il le faut, en suspendant l'exécution du programme principal, qui en fait ne se rend pas compte qu'il est interrompu. La routine d'interruption est à l'adresse 4. Le programme principal doit sauter par dessus..

Prenons un exemple trop simple. Le programme principal compte de façon visible sur le portC, toutes les 0.1s par exemple. L'interruption du timer remet à zéro ce compteur toutes les secondes.

En plus de l'initialisation du port qui compte et du timer, il faut initialiser l'interruption, ce qui veut dire activer 2 bits dans le registre IntCon, une pour une activation générale des interruptions (GIE General Interrupt Enable) et une pour l'interruption spécifique du Timer0 (TOIE Timer0 Interrupt Enable).

Ce qu'il faut faire à chaque interruption due à la bascule TOIF, c'est de quittancer le TOIF, et utiliser un compteur auxiliaire pour effacer le PortC toutes les secondes (on ne peut pas ralentir le timer suffisamment).

Le cœur de la routine d'interruption est donc

```
; On désactive le bit qui a demandé l'interruption
Clr  Intcon:#TOIF
Move #IniTmr0,W
Move W,TMR0
Inc  CInt ; Compteur auxiliaire par 256 pour ralentir
Skip,NE
Clr  PortC ; Chaque 256 x 256 x 16 us = ~1s
```

On voit que ce programme, qui va être appelé sans avertir quand le timer déborde, utilise le registre W. Il ne faudrait pas que le registre W, fréquemment utilisé par le programme, soit modifié par l'interruption. On doit donc rajouter 3 instructions au début et 5 instructions à la fin pour que la routine d'interruption sauve et rétablisse W, ainsi que les flags Z et C.

Le programme **T877T0Int.asm**, est le suivant :

<pre>\prog;T877T0Int.asm Timer par interruption ; Le programme principal compte à ~15Hz sur le ; PortC et l'interruption mets à zéro ce portC toutes les 256 boucles d'interrupt. ; On voit que le compteur ne vas jamais très loin. ;Agir sur IniOption et sur IniTmr0 .Proc 16F877 .Ref 16F877 \var;Registres SaveF = 16'20 SaveW = 16'21 CInt = 16'22</pre>	<pre>.loc 0 Jump Deb .loc 4 ; Interrupt tous les 256x16 us Move W,SaveW; Ne modifie pas F Swap F,W ; Truc Move W,SaveF ; On désactive le bit qui a demandé l'interruption Clr Intcon:#TOIF Move #IniTmr0,W Move W,TMR0 Inc CInt ; Compteur auxiliaire par 256 pour ralentir Skip,NE Clr PortC ; Chaque 256 x 256</pre>	<pre>\b;Programme principal Deb: Set Status:#RP0 Move #DirC,W Move W,PortC Clr Status:#RP0 Move #IniOption,W ; Prescaler :16, 16 us Move W,Option Clr IntCon:#TOIF Move #2**GIE+2**TOIE,W Move W,IntCon Loop: Inc PortC ; Attente pour incrémenter env 15 fois par seconde</pre>
--	---	---

<pre> CX1 = 16'23 CX2 = 16'24 \var;Ports \b;PortC comme compteur DirC = 0 \const; Initialisation IniOption = 2'0000001; :16 IniTmr0 = -200 ; 256-200 </pre>	<pre> x 16 us = ~1s F\$: Swap SaveF,W Move W,F Swap SaveW Swap SaveW,W RetI </pre>	<pre> A\$: DecSkip,EQ CX1 Jump A\$ DecSkip,EQ CX2 Jump A\$ Jump Loop .Loc 16'2007 .16 16'3F39 .End </pre>
---	--	--

Rappelons que le TMR0 est un compteur. Si on veut que TOIF s'active après un comptage de 200, il ne faut pas l'initialier avec la valeur 200 (il augmenterait de 200 à 256=0 et TOIF s'activerait), mais avec la valeur 256-200, identique pour l'assembleur à -200.

4.4 Timer1

Le timer 1, quand il est disponible comme sur le 16F877, est un compteur 16 bits avec prédiviseur. Le programme T877T1Int.asm teste son fonctionnement. Il peut être couplé à 2 pins sur lesquelles on branche un quartz horloger à 32 Khz, ce qui permet de programmer une horloge précise..

Le timer2 est beaucoup plus riche et permet du PWM (voir section 6.2) et une mesure plus efficace de durées d'impulsions.

5. Entrées analogiques

Les PICs n'ont pas tous des canaux analogiques. Si oui, ils ont la même structure avec une précision de 10 bits, et nous utiliserons le mode qui laisse tomber les 2 bits de poids faible : il faut une construction qui respecte plusieurs règles pour avoir des mesures fiables en 10 bits. Le document de Bigonoff traduit de façon très claire toutes les possibilités offertes par Microchip <http://www.abcelectronique.com/bigonoff/>. Nous allons simplifier au maximum.

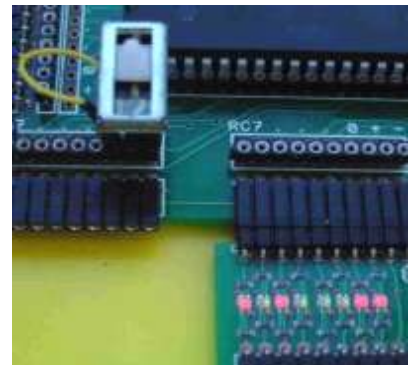
Si le PIC n'a pas d'entrées analogiques, on peut lire très efficacement des signaux analogiques en mesurant des temps de charge et décharge de condensateurs. Voir <http://www.didel.com/picg/doc/DopicAD.pdf>

5.1 Initialisation

Le 16F877 a 8 canaux analogiques possibles sur le portA et le portE. Il en suffit souvent d'un ou deux, et on veut garder les autres lignes pour des entrées-sorties tout-ou-rien. Sur le 16F877, 4 bits du port AdCon0 offrent quelques combinaisons, et il faut bien réfléchir pour câbler son application. Les processeurs plus récents offrent une plus grande flexibilité.

Il faut comprendre que dans l'initialisation, on dit quels sont les canaux analogiques en service (registre AdCon1), mais le programme lit un canal à la fois et il faut dire lequel on lit (registre AdCon0, nommé AnSel sur d'autres PICs).

Pour la lecture, il faut sélectionner le canal, attendre 20 microsecondes que les aiguillages internes se positionnent, démarrer la conversion et attendre qu'elle soit terminée, avec le résultat dans AdResH (on a décidé d'oublier la partie poids faibles AdResL, mais on peut la lire sur les bits 7 et 6).



Par exemple, on veut initialiser seulement RA0 en entrée analogique et la lire.

Dans les définitions on va déclarer

```

DirA = 2'010001 ; ou 2'111111
IniAdCon0 = 2'00001110 ; sel RA0 seul
SelAd0 = 2'11000001; sel AD0 en mode8 bits

```

Dans l'initialisation on doit avoir

```

Clr   PortA
Set   Status:#RP0
Move  #IniAdCon1,W
Move  W,AdCon1
Move  #DirA,W
Move  W,PortA
Move  #0,W
Move  W,TrisC

```



```
Clr Status:#RP0
```

Et dans le programme

```
Move #SelAd0,W
Move W,AdCon0
Call Del20 ; attente 20 us
Set AdCon0:#Go ; C'est parti
C$: TestSkip,BC AdCon0:#Go
Jump C$ ; On attend
Move AdResH,W
Move W,PortC ; On affiche la valeur
```

Pour le test, il faut brancher un potentiomètre avec le point milieu sur RA0 et utiliser le programme **T877Ana.asm**

Sans potentiomètre, si on touche avec le doigt, on perturbe l'entrée à haute impédance.

5.2 Lecture par interruption

L'attente sur la conversion prend 100 à 200 microseconde. On a donc avantage à lire les canaux analogiques par interruption. Passez plus loin si cela semble compliqué.

Le timer déclenche toutes les 20ms une interruption qui démarre une cascade d'interruptions pour lire les canaux sur RA0, RA1, RA3 (RA2 est spécial).

Au moment de l'interruption, il faut décider d'où elle vient, et un pointeur doit mettre les valeurs lues dans une zone mémoire.

Dans les définitions, il faut déclarer

```
IniOption = 2'00000110 ; divise par 128
IniTmr0 = -156 ; 156 x 128us = 20ms
IniAdCon310 = 2'00000100 ; sel RA3 RA1 RA0
SelAd0 = 2'11000001; sel AD0 en mode8 bits
SelAd1 = 2'11001001
SelAd2 = 2'11010001
```

Dans les variables, il faut déclarer

```
Var SaveF
Var SaveW
Var NCanal
Var Canal0
Var Canal1
Var Canal2
```

L'initialisation contient les instructions

```
Clr PortA
Set Status:#RP0
Move #TrisA,W
Move W,PortA
Move #TrisC,W
Move W,PortC ; Affichage
Move #IniAdCon310,W
Move W,AdCon1
Clr Status:#RP0
Move #IniOption, ; Le timer
Move W,Option
Clr IntCon:#TOIF
Move #2**GIE+2**PEIE+2**TOIE,W ; Timer seul
Move W,IntCon
```

La routine d'interruption contient les instructions

```
TestSkip,BC IntCon :#TOIF
Jump InterTimer
TestSkip,BC PIR1 :#ADIF
Jump InterAna
Jump Error
```

InterTimer :

```
Clr IntCon:#TOIF
Move #IniTmr0,W
Move W,TMR0
Clr IntCon:#TOIE ; Coupe les interrupt timer
```

; On met en route 3 interrupt Ana

```
Move #3,W
Move W,NCanal
Move #SelAd0,W
Move W,SaveSelAD
Set PIE1:#ADIE
```

DebAna:

```
Clr PIR1 :#ADIF
```

```

Move SaveSelAD,W
Inc SaveSelAD
Move W,AdCon0
Move #2'00001000,W
Add W,SaveSelAD
Call Del20
Set AdCon0:#Go ; C'est parti
Jump Rt ; Suite au prochain interrupt Ana
InterAna : ; AdCon0:#Go a déclanché l'interrupt
; Il faut sauver AdResL au bon endroit
Move Canal0,W
Move W,FSR
Move NCanal,W
Add W,FSR
Move AdResH,W ; mets à zéro Go
Move W,{FSR} ; voir section 6 ??
; on prepare la lecture suivante
DecSkip,EQ NCanal
Jump Rt
; On a lu les 3 canaux, on repasse in interrupt timer
Clr PIE1:#ADIE
Set Intcon :#TOIE
Rt :

```

Le programme complet se trouve sous **T877AnaInt.asm**

On voit que gérer plusieurs évènements par interruption n'est pas très simple. Pour une télécommande infrarouge qui doit lire trois potentiomètres et envoyer un train d'impulsions toutes les 40 us, les interruptions ne sont pas utilisées car les deux tâches à effectuer ne sont pas simultanées. L'écriture du programme est très simple.

6 Commande de moteurs

Il existe plusieurs types de moteurs de puissance très variée. Seuls les moteurs de faible puissance nous intéressent. La commande en tout-ou-rien d'un moteur ou électro-aimant s'apparente à la commande d'une LED, avec éventuellement un transistor amplificateur, donc rien de plus à dire, si ce n'est que les pointes de courant au démarrage du moteur peuvent perturber le fonctionnement du processeur. On vérifie en remplaçant les moteurs par des LEDs.

6.1 Moteurs bidirectionnels

Les moteurs à courant continu ont deux fils et se commandent comme des diodes bicolores, mais en général aux travers d'amplificateurs appelés "ponts en H".

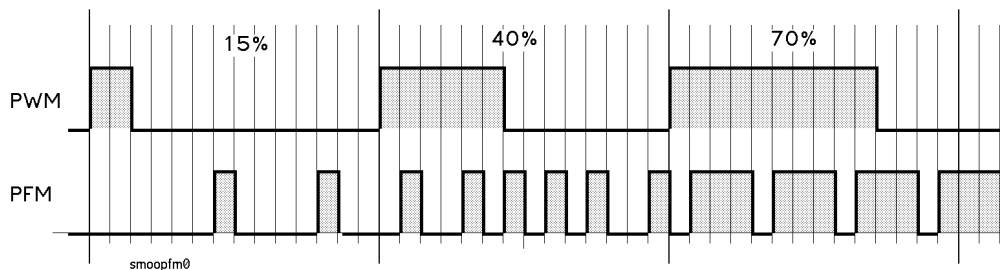
Si le courant demandé est entre 10 et 50mA, on peut lier des sorties du processeur entre elles pour augmenter le courant. Microchip parle de 20mA par sortie, mais c'est en court-circuit !

6.2 PWM (Pulse Width Modulation)

Le principe du PWM est de varier la largeur d'une impulsion répétitive. Les processeurs performants ont des canaux PWM obtenus en programmant le timer 2 ou le timer3, mais il faut utiliser les sorties prévues sur le microcontrôleur. Ce n'est pas très simple si le sens du moteur doit être inversé. Il faut bien lire la documentation du fabricant et passer du temps sur le programme de test.

En logiciel, le principe est simple et efficace, et permet de commander autant de moteurs que l'on veut, sur les sorties que l'on veut, à une fréquence et résolution adéquates pour des application non industrielles.

Les explications générales sont données sous www.didel.com/picq/picq87x/Picq75.pdf page 28. Le programme de test **T877Pwm.asm** montre comment commander un moteur unidirectionnel et un moteur bidirectionnel en logiciel. Le programme de test **T877PwmT2.asm** utilise le Timer2.



6.3 PFM (Pulse Frequency Modulation)

Le PFM est mal connu car il n'est pas utilisé sur les moteurs performants. Mais pour les moteurs jouet ou miniatures, il permet une commande à faible vitesse très intéressante. La fréquence est faible et facilite la programmation multi-tâche.

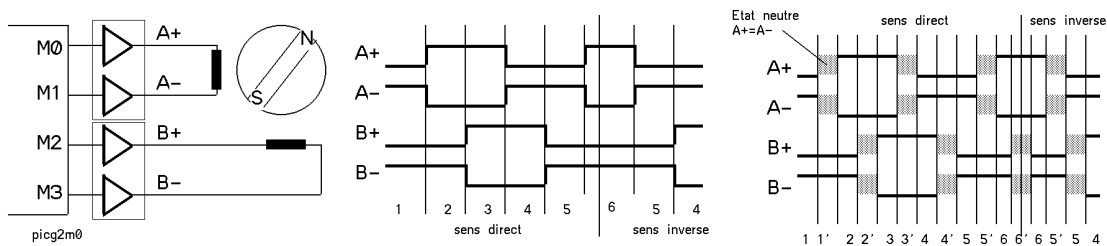
Les explications sont données sous www.didel.com/picg/picg87x/Picg75.pdf page 29.

Le programme de test **T877Pfm.asm** montre comment commander un moteur unidirectionnel et un moteur bidirectionnel en logiciel. Pour des moteurs "pager" qui ont une tension de démarrage importante à cause du frottement, le PFM avec des impulsions de 2 à 5ms suffisantes pour faire "décoller" le moteur permettent de couvrir toute la gamme de vitesses.

6.4 Moteur pas-à-pas

L'avantage du moteur pas à pas est que son angle de rotation dépend du nombre de pas, tant que le couple maximum n'est pas dépassé.

Ces moteurs sont commandés par 4 phases, avec des demi-phases possibles pour un mouvement plus régulier. Une table définit la séquence. Pour changer de sens on déplace le pointeur en sens inverse, en gérant correctement le passage par les extrémités. Des explications générales se trouvent sous www.didel.com/picg/picg87x/Picg75.pdf page 27, mais les exemples de programmes concernent des moteurs à 6 phases. La transposition est facile.



Les moteurs horlogers (Lavet) ont 6 phases (Switec, Wellgain). Des explications détaillées sont données dans le lien ci-dessus et sous www.didel.com/picg/doc/DopiSwi.pdf. Une doc plus récente avec exemples se trouve sous www.didel.com/bot/step/Step.doc

Avec du PWM ou du PFM, on peut lisser les phases pour se rapprocher d'une excitation sinusoïdale. Ceci ne présente pas d'intérêt, mais c'était une recherche intéressante !

<http://www.didel.com/picg/doc/DopiSmoo.pdf> et sous

<http://www.didel.com/picg/doc/DocLimot.pdf>

6.5 Moteurs "brushless"

Les moteurs sans collecteur sont des moteurs pas à pas asservis par une électronique qui permet d'obtenir le couple maximum. Ils sont donc commandés par des circuits extérieurs au microcontrôleur qui reçoivent les ordres de vitesse via une interface parallèle ou série (les modélistes avion utilisent des circuits avec une entrée PPM comme les servos).

6.6 Servos de télécommande et PPM

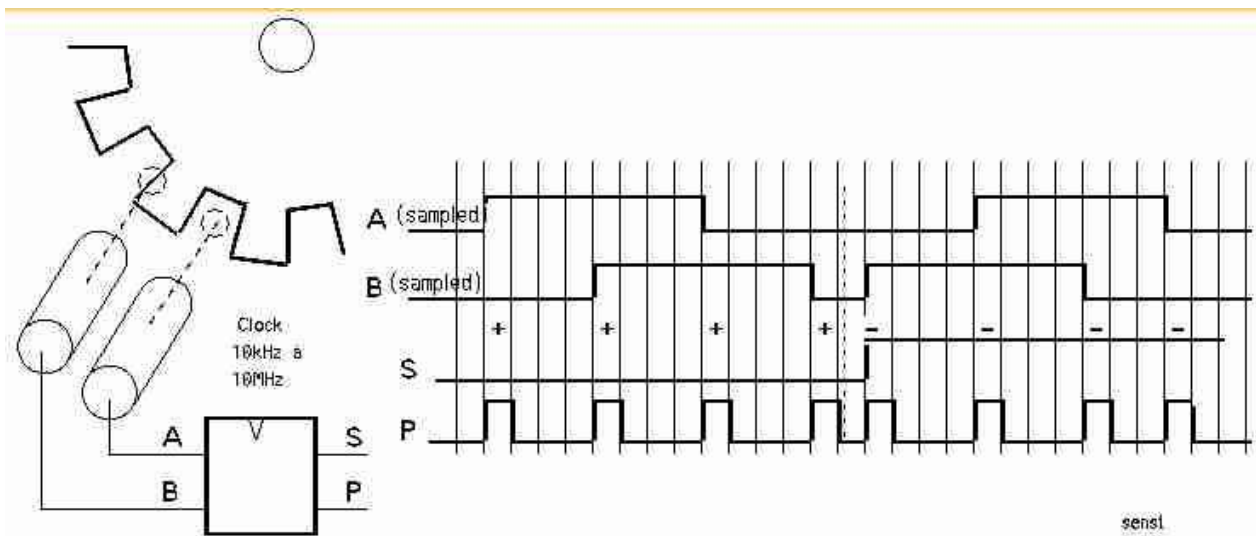
Les servos de télécommande ont une entrée qui est une impulsion de 1 à 2ms répétée toutes les 20ms. La durée de l'impulsion fixe la position du servo (Pulse Position Modulation).

Pour un exemple de programme de codage simple (monotâche), voir

<http://www.didel.com/picg/doc/DocServo.pdf>.

6.7 Encodeur

Pour connaître la position d'un moteur continu ou d'un axe, on utilise un encodeur (quadrature encoder) qui génère des impulsions déphasées. A défaut de circuits spécialisés, un programme échantillonne et compte/décompte les incréments jusqu'à des périodes de 100 us par transition. Des algorithmes efficaces sont expliqués sous <http://www.didel.com/picg/doc/PicSoft.pdf> (page 15-16) et <http://www.didel.com/picg/picg87x/Picg75.pdf> (page 29)



6.8 Capteurs de distance et autres

Les capteurs génèrent des impulsions ou des niveaux analogiques que les microcontrôleurs gèrent facilement. Les capteurs intelligents ont un contrôleur propre et communiquent en général en série. Une documentation sur les capteurs de distance par infrarouge se trouve sous <http://www.didel.com/doc/sens/Doclr.pdf>
<http://www.didel.com/doc/sens/Doclrt.pdf>
<http://www.didel.com/doc/sens/DocSharp.pdf>

Internet permet de trouver beaucoup d'information plus ou moins complète sur les capteurs et leurs interfaces.

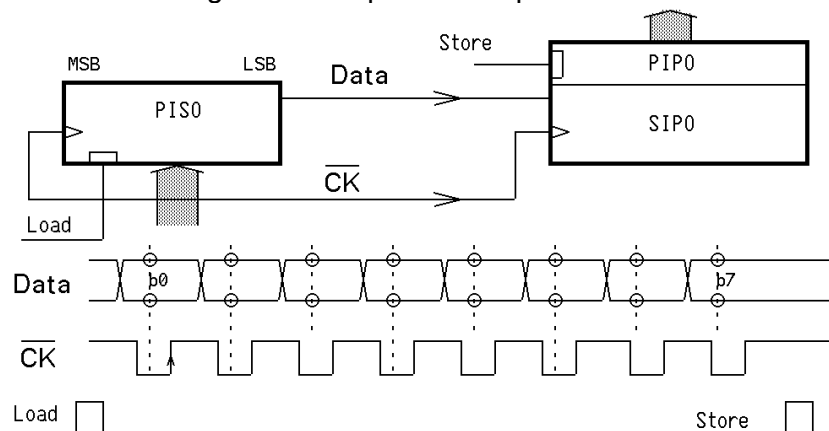
Les capteurs «intelligents», c'est à dire qui ont un préconditionnement des signaux, envoient souvent l'information en série vers le processeur.

7 Transferts série

Si une application comme un affichage a besoin de plusieurs lignes de commande, il est préférable d'utiliser un circuit spécialisé et lui transférer l'information en série sur quelques fils, donc en n'utilisant que 1 à 4 sorties du processeur.

7.1 Registres série

L'information d'un registre peut être décalée en série. A chaque impulsion d'horloge, un bit est décalé. Une impulsion Load charge au début l'information dans le registre série (PISO – Parallel In Serial Out), 8 impulsions CK décalent. Si le registre destination (SIPO – Serial In Parallel Out) a un registre en sortie une impulsion Store charge ce registre après les 8 impulsions de décalage. La polarité de l'horloge et des impulsions dépend des circuits utilisés.



Le microcontrôleur peut être d'un côté ou de l'autre. L'instruction de décalage va simuler le registre et les signaux nécessaires seront activés/désactivés par programmation.

Les routines d'écriture et de lecture sont données sous www.didel.com/picg/picg87x/Picg76.pdf page 36. Le circuit Ext8i8o ajoute 8 entrées et 8 sorties en n'utilisant que trois lignes du processeur www.bricobot/kits/Ext8i8o.pdf Les signaux Load et Store sont générés à l'intérieur du module. Le programme `T877Ext8i8o.asm` teste ce module d'extension.

Notons encore que les registres à décalage peuvent être cascades. Avec le même nombre de lignes utilisées sur le microcontrôleur, on peut commander autant d'entrées sorties que l'on veut, mais évidemment cela prend du temps de transfert.

7.2 Transfert en écriture

La routine exemple ci-dessous doit être adaptée à l'application. Le décalage ne doit pas nécessairement de faire à droite, et la polarité des impulsions peut être inversée.

```
\rout:Write|Transfer 8 bits serially
\in:DataOut, transferred LSB first
\mod:DataOut, C1
```

Write:

```
Move #8,W
Move W,CntCk
```

L\$:

```
RRC DataOut
Skip,CC
Set PortA:#bData
Skip,CS
Clr PortA:#bData
Clr PortA:#bCk
Set PortA:#bCk
DecSkip,EQ CcntCk
Jump L$
Set PortA:#bStore
Clr PortA:#bStore
Ret
```

7.3 Transfert en lecture

```
\rout:Read|Get 8 bits serially
\out:DataIn read, transferred LSB first
\mod:DataIn, CntCk
```

Read:

```
Move #8,W
Move W,CntCk
Set PortA:#bLoad
Clr PortA:#bLoad
```

L\$:

```
ClrC
TestSkip,BC PortA:#bData
SetC
RRC DataIn
Clr PortA:#bData
Clr PortA:#bCk
DecSkip,EQ CntCk
Jump L$
Ret
```

7.4 SPI

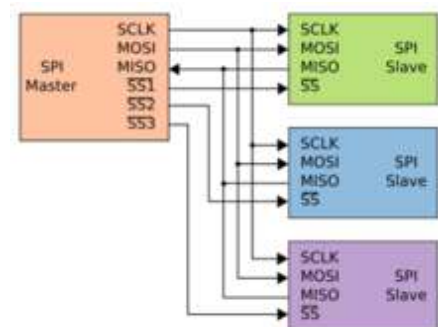
Le SPI utilise 3 lignes pour le transfert et une ligne pour la sélection de chaque esclave.

http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Une logique interne envoie et reçoit l'information et active un flag quand le transfert est terminé. Par interruption, des débits importants, par exemple pour commander un affichage graphique, sont obtenus, mais les PICs sont rarement adaptés pour ce type d'application.

Voir aussi la doc plus ancienne

www.didel.com/picg/doc/DocSPI.pdf



7.5 I2C

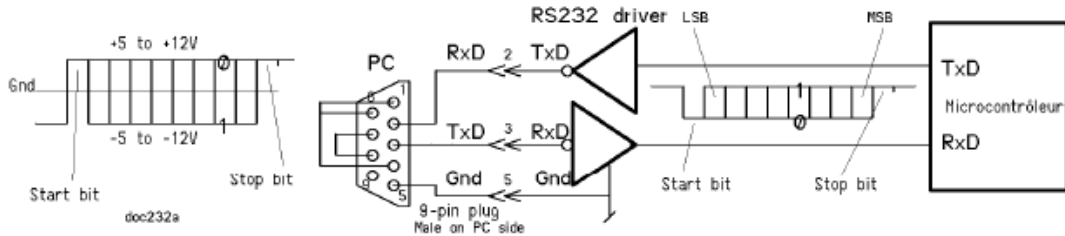
I2C est souvent intéressant pour se relier à des circuits horloge ou mémoire de stockage d'information. Les PICs haut de gamme ont des circuits câblés pour gérer le I2C, mais pour des applications simples et lentes, les routines logicielles sont tout aussi faciles à maîtriser et prennent une centaine d'octets.

Pour plus de détails, voir www.didel.com/picg/doc/DopicI2C.pdf.

7.6 USART

Tous les PC avaient anciennement une entrée dite "RS232" très pratique pour communiquer avec le microcontrôleur. Voir www.didel.com/dev877/PdSerie.doc comme exemple. Avec un adaptateur USB-RS232 on obtient théoriquement le même service. Côté PC, un programme "terminal" dialogue avec le clavier-écran via la ligne série.

Une fois la liaison établie, il faut installer un programme de communication série. Hyperterminal est difficile à configurer. TeraTerm est facile d'emploi http://en.wikipedia.org/wiki/Tera_Term Il peut se télécharger depuis www.didel.com/dev877/Ttermpro.zip.



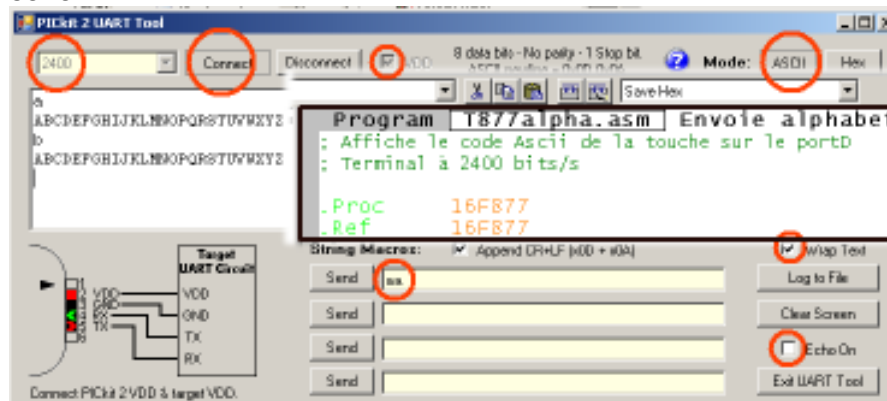
Tous les microcontrôleurs PIC qui ont un USART (Universal Serial Asynchronous Receiver and Transmitter) câblé semblent avoir les mêmes registres de commande. Il faut définir la vitesse de transmission et gérer le protocole qui est simple si on ne se préoccupe pas de la gestion des erreurs de transmission. Pour envoyer, on écrit dans le registre TxD et on surveille le bit TxIF pour savoir si on peut envoyer le byte suivant.

Si on attend de l'information, on surveille un bit RclF qui s'active quand un byte est arrivé. La lecture dans le registre RcReg met à zéro RclF.

Le programme **T877Alpha.asm** envoie l'alphabet sur l'écran en réponse à une touche pressée. Le transfert se fait à 2400 bits/s, ce que l'on peut facilement changer par programmation (à la fois dans l'initialisation et dans l'USART tool).

Le document <http://www.didel.com/picg/doc/DopicSer.pdf> donne des routines pour contrôler sans interface série et avec.

A noter encore que le Pickit2 a un mode convertisseur série-UART (menu Tools, déplacer le Pickit). Le microdual 2840, avec sauf erreur tous les PICs 28 et 40 broches, peut donc être utilisé très facilement pour développer des applications avec communication série.



On peut taper sur le clavier, envoyer une chaîne de caractères.

On peut imaginer des applications où le microcontrôleur est un esclave du PC, répond à des ordres, donne l'état de ses capteurs. L'application principale tourne dans le PC et n'est pas limitée en complexité.

7.7 USART par logiciel

Programmer un transfert série compatible RS232 est facile, mais il faut respecter un timing précis, et ne pas avoir d'interruption pendant le transfert. La vitesse de transfert est limitée à 2400 bits/s avec un processeur à 4 MHz et les routines nécessaires utilisent moins de 50 instructions. Voir www.didel.com/picg/doc/DopicUti.pdf qui documente aussi des routines pour afficher des textes, valables dans tous les cas.

7.8 Transferts "one-wire" Dallas

Dallas propose des circuits commandés par 2 fils seulement : la masse (gnd) et l'alimentation qui est pulsée pour transférer une information bien assez rapide pour beaucoup d'applications. Pour plus de détails, voir <http://www.maxim-ic.com/products/1-wire/>

7.9 Petra

Le Bus Petra de Didel permet d'adresser 16 unités qui ont leur propre processeur, et d'avoir dans 16 variables successives l'image des informations gérées par ces unités (par exemple une mesure de distance).

Pour plus de détails, voir <http://www.didel.com/Petra.pdf>

7.10 Autres bus

Industriellement, le Can-bus est très utilisé, et les microcontrôleurs haut de gamme le supportent. Pour les petites applications, on peut faire beaucoup avec des registres à décalage. Pour communiquer entre microcontrôleur, ce que fait Petra, il faut une programmation très attentive concernant les durées d'exécution.

8 EeProm et bootloader

Seuls quelques PICs en boîtier 6 et 8 pattes n'ont pas de mémoire Eeprom. Cette mémoire de 64 à 256 octets peut être écrite et lue par le programme.

Quelques processeurs 28 et 40 pattes, dont le 16F877, ont une possibilité supplémentaire de lire et modifier la mémoire programme, dite mémoire Flash, ce qui permet de lire ou écrire des grandes tables, et charger un programme sans passer par le programmeur, mais via une ligne série.

8.1 Mémoire EeProm

La lecture de la mémoire EeProm se fait en initialisant une variable adresse EeAdr et en lisant l'information dans la variable donnée EeData. Ces variables ne sont pas dans la banque 0 avec les ports, et il faut changer souvent de banque.

L'écriture est tout aussi simple, mais un groupe d'instructions bizarres doit être ajouté pour se protéger d'une écriture intempestive si le programme déraile et l'écriture dure quelques ms. Le bit WR du registre EECON1 en banque 3 est actif pendant que l'écriture se fait.

Le programme de test **T877EeProm.asm** contient les macros et routines que l'on peut directement utiliser dans ses programmes, s'ils n'utilisent pas l'interruption.

8.2 Mémoire Flash

La mémoire programme du 16F877 peut être lue et écrite. La lecture est facile comme pour l'EeProm. Toutefois, le codage de l'adresse et des données se fait sur des paires de variables, puisque la mémoire peut avoir 4k (adresse 2 bits) et les instructions ont 14 bits.

L'écriture se fait instruction par instruction avec le 16F877, mais pour le 16F877A et le 16F882/884 l'écriture se fait par groupes de 4 positions consécutives pour gagner du temps, ce qui pose de jolis problèmes d'alignements. La routine a été développée pour le bootloader du Dev877 et est à disposition.

8.3 Boot loader

L'intérêt de l'écriture en mémoire programme est que l'on peut avoir dans une zone réservée du processeur qui ne sera jamais modifiée, un programme appelé "boot loader" qui charge depuis l'interface série le programme à exécuter. Ceci évite un programmeur de PIC, sauf naturellement pour mettre le boot loader en mémoire. Avec le Pickit2, on a le programmeur et l'interface série dans la même unité.

SmileNG peut être configuré pour transmettre en série le .hex à la fin de l'assemblage, si on sait initialiser le bon COM port. Le Didelbot www.didel.com/bot/ et la carte Dev877

www.didel.com/07dev877/indexF.html utilisent cette facilité, qui est prévue dans le nouveau Wellbot.

9 Tables

Les PICs ont une instruction très efficace pour gérer des tables. Cette instruction ajoute une valeur au compteur d'adresse, mais attention, l'addition se fait sur 8 bits, donc on reste dans ce que l'on appelle une page mémoire 8 bits. Le compteur d'adresse PC (qui pointe les instructions) est coupé en deux : PCLatH pour les poids forts (5 bits), PCL pour les poids faibles.

Pour nos applications, travaillons en page 0, avec PCH=0, et toutes les tables en page 0. Le programme, les interruptions, les routines n'ont pas de contraintes et peuvent se trouver après les tables. Les programmes commencent par

```
.Loc 0
    Jump Debut
.Loc 4
    Jump Inter
; Tables longueur max 252 octets.
```

9.1 Tables de saut

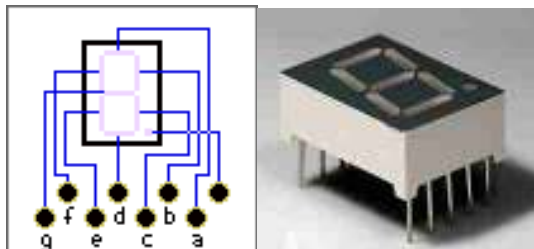
Un joli exemple considère 3 interrupteurs qui selon leur position définissent 8 comportements, conditions initiales, etc. Ces interrupteurs sont câblés sur les bits 2,1,0 du port C par exemple.

```
Loop: Move PortC,W
      And #2'111,W
      Add W,PCL ; on saute 0,1,..7 pos plus loin selon W
      Jump Do0
      Jump Do1
      ...
      Jump Do7
Do0:  ... ce qu'il faut faire si tous les interrupteurs à 0
      Jump Loop
Do1:
      Etc.
```

9.2 Tables de conversion

Les tables de conversion utilisent une autre instruction particulière du PIC, l'instruction `RetMove #Val,W`. Comme son nom l'indique, c'est un RET, retour de routine, qui en même temps met la valeur dans W. On écrit alors une routine d'aiguillage qui a autant de Return que nécessaire. Par exemple, si on a besoin de commander un affichage 7 segments, l'entrée dans la table est une valeur de 0 à 16'F dans W, et la sortie dans W est immédiatement utilisée pour allumer les segments.

```
\in: W chiffre 4 bits
\out: W code pour allumer les segments
GetSegments : ; -gfedcba for bits 7..0
Add W,PCL
RetMove #2'00111111,W ; 0
RetMove #2'00000011,W ; 1
...
RetMove #2'01111000,W ; F
```



La partie de programme qui affiche les segments est

```
Move Chiffre,W ; Variable qui contient un nombre de 0 à 9
Call GetSegments
Move W,PortC
```

Si Chiffre contient un nombre supérieur à 16'F, l'affichage dépend des instructions qui suivent. Pour éviter ce risque, on commencera la routine `GetSegments` par

```
Move Chiffre,W
And #2'00001111,W
Add W,PCL
```

9.3 Astuces

Les tables d'évolution pour un moteur pas à pas se font tout naturellement avec cette technique, avec une astuce supplémentaire si l'on met des instructions dans la table :

```
AvancePas :
    Move Pos,W ; le compteur par 6 qui définit l'excitation du moteur
    Inc Pos
    Add W,PCL
    RetMove #2'01111111,W
RetMove #2'01111111,W
RetMove #2'01111111,W
RetMove #2'01111111,W
RetMove #2'01111111,W
    Clr Pos
RetMove #2'01111111,W
```


On voit qu'après le 6^e pas, on recommence, sans avoir eu besoin de comparer si on était arrivé à 6, ce qui aurait coûté quelques instructions.

9.4 Protection

Une table mal placée ou mal accédée peut créer des plantées difficiles à analyser.

Premier conseil : saturer le pointeur pour rester dans la table..

Par exemple, si le pointeur dans la table est Ptable, et la longueur MaxPtable, on écrit

GetTable:

```
Move #MaxPtable,W
Sub W,Ptable
Skip,CC
Clr Ptable
Add Ptable,W ; c'est saturé
Add W,PCL
RetMove # ...,W
...
```

Deuxième conseil : pour être sûr que l'on ne déborde pas de la page, mettre un .If qui teste le compteur d'adresse. Pour un message d'erreur si on n'est plus en page 0 :

```
.If APC/256.NE.0
On déborde (pas de ; pour qu'un message d'erreur apparaisse
.Endif
```

9.5 Calculs

Les tables sont un outil puissant pour éviter des calculs, les applications visées ayant en général besoin de peu de précision. Si on a une conversion à faire sur une valeur qui a une précision de 3%, on ramène cette valeur entre 0 et 31 et on passe par une table pour avoir la valeur correspondante. Par exemple, 32 vitesses sont bien suffisantes pour un robot, mais on peut être intéressé à avoir une réponse logarithmique avec des vitesses très lentes. Une table donne les valeurs PFM, signées si nécessaire.

10 Accès indirect

Un Move par exemple déplace une valeur qui peut être immédiate (#Val), directe (Reg) ou indirecte, c'est à dire dans ce 3^e cas que l'opérande contient l'adresse d'une variable qui est l'adresse.

Vous avez gagné 100.- pour un concours

On vous donne les 100.- → adressage immédiat

On vous donne l'endroit où aller les chercher → adressage direct

On vous dit d'aller à tel endroit ou on vous dira où aller → adressage indirect (pratique si ce n'est pas toujours au même endroit)

Les PICs sont très primitifs avec un seul registre indirect, appelé FSR (File Select Register).

Par exemple, si une variable est à l'adresse Toto = 16'43, on peut lire son contenu de deux façons :

```
Move Toto,W ; Adressage direct
```

ou

```
Move #Toto,W
Move W,FSR ; FSR connaît l'adresse de notre variable
Move {FSR},W ; Adressage indirect
```

L'intérêt est que si on veut maintenant lire le contenu de l'adresse qui suit en mémoire, il suffit d'augmenter FSR de 1. On peut ainsi dans une boucle, parcourir une table de variables.

10.1 Effacer une zone mémoire

Dans l'initialisation de tout programme, il est conseillé de mettre toutes les variables à zéro, même celles qui ne sont pas utilisées. Cela prend peu de temps, et on ne risque pas de démarrer avec des variables qui n'ont pas la même valeur à chaque enclenchement.

La variable en première position de la zone des variables est utilisée comme décompteur.

Le fichier de référence inséré au début de chaque programme (.Ref 16F877) connaît le début et la fin de la zone des variables.

```
\b; On efface toutes les variables - 500 us
```

```
Move #DebVar+1,W
Move W,FSR
Move #FinVar-DebVar-1,W
```

```

Move W, DebVar
L$: Clr {FSR}
Inc FSR
DecSkip, EQ DebVar
Jump L$

```

S'il y a des variables dans une 2^e banque, il faut aussi les effacer.

10.2 Besoin de 2 pointeurs ?

Si on doit gérer un pointeur dans la routine d'interruption et un autre dans le programme principal, il faut sauver en mémoire les valeurs des deux pointeurs et commuter lorsque c'est nécessaire :

```

Move FSR,W
Move W, SaveFSR1
Move SaveFSR2,W
Move W, FSR

```

On donnera naturellement des noms plus explicites à SaveFSR1 et SaveFSR2.

11 Séquencement et multitâche

On doit souvent faire plusieurs choses en même temps, et chaque chose se fait en plusieurs étapes successives. Par exemple, si on attend un signal infrarouge tout en commandant la vitesse des moteurs, la tâche infrarouge est surveillée et un compteur suit l'évolution.

Si on doit surveiller un ascenseur, une tâche surveille les demandes et une tâche s'occupe du déplacement de l'ascenseur. Il y a beaucoup de façons de procéder, selon les applications.

11.1 Exemple 1 – Clignotement à vitesse variable

Prenons un cas simple avec deux tâches, une qui clignote une LED, et une qui surveille un poussoir pour modifier la vitesse de clignotement.

Toutes les 20ms puisque les tâches sont très lentes, on s'occupe successivement des deux tâches, qui partagent la variable VitCligno, qui définit le temps entre les impulsions lumineuses.

Clignotement : La routine TaskCli a deux sous-tâches définies par le compteur TkCli.

- Allumer la Led pendant 0.1s (IniLedOn=5 cycles compté par la variable CntLed)
- Eteindre pendant VitCligno cycles compté par la variable CntLed

Poussoir : La routine TaskPous a trois sous-tâches définies par le compteur TkPou

- Attendre que l'on pèse
- Attendre que l'on relâche en mesurant la durée
- Agir sur la variable VitCligno

Détaillons les instructions nécessaires pour ces tâches.

```

Tkcl1: Clr PortC:#bLed ; allumé
      DecSkip, EQ CntLed
      Ret ; on reste dans la même tâche
; On prépare la tâche suivante
      Move VitCligno,W
      Move W,CntLed
      Inc TkCli
      Ret
Tkcl2: Set PortC:#bLed ; éteint
      DecSkip, EQ CntLed
      Ret ; on reste dans la même tâche
; On prépare la tâche suivante
      Move #IniLedOn,W
      Move W,CntLed
      Clr TkCli
      Ret

```

Pour la tâche poussoir, on veut distinguer les actions courtes, qui vont augmenter la fréquence de clignotement, et les actions longues, qui vont diminuer cette fréquence.

Ceci peut se faire avec un décompteur de durée de l'action, et le truc de saturer le compteur à zéro. On initialise le compteur pour qu'il arrive à zéro au bout d'une seconde (IniCntPous = 50 puisque 50x20ms = 1s)

```

TkPou1: TestSkip, BS PortC :#bPous
      Ret ; on attend
; Action sur le poussoir, on prépare la tâche
      Move #IniCntPous,W
      Move W,CntPous

```

```

        Inc    TkPous
        Ret
TkPou2:  Dec  CntPous
        Skip,NE
        Inc    CntPous      ; Sature à 1
        TestSkip,BC  PortC :#bPous
        Ret    ; on attend le relâchement
; On décide si on augmente ou diminue VitCligno
        DecSkip,EQ  CntPous
        Jump  Augmente
Diminue: Dec    VitCligno
        Skip,NE
Augmente: Inc    VitCligno  ; sature à 1
        Clr    TkPous
Ret

```

Le programme complet doit définir les constantes et les variables, initialiser les pointeurs de tâche à zéro et appeler dans une boucle synchronisée par le timer à 20ms les deux tâches.

```

Loop:  TestSkip,BS  Intcon:#TOIF
        Jump  Loop  ; On attend 20ms
        Clr    Intcon:#TOIF
        Move  #IniTmr0,W
        Move  W,TMR0      ; Réinit les 20ms
        Call  DoTaskCli
        Call  DoTaskPous
        Jump  Loop

DoTaskCli:
        Move  TkCli,W
        Add   W,PCL
        Jump  TkCli1
        Jump  TkCli2

DoTaskPous:
        Move  TkPou,W
        Add   W,PCL
        Jump  TkPou1
        Jump  TkPou2

```

Suivent les tâches décrites plus haut. Le programme complet est sous **T877TaskCli.asm**

11.2 Exemple 2 – Robot évitant les obstacles

Un robot a deux capteurs de distance en tout ou rien ou deux moustaches. On veut programmer un évitement d'obstacles où le robot va reculer et tourner pour éviter au mieux l'obstacle.

Il faut donc se souvenir de quelle moustache a touché jusqu'à ce que la manœuvre d'évitement soit terminée. Le robot est de type voiture avec une variable Vitesse qui définit le PFM du moteur et une variable Virage en tout ou rien.

Il faut bien se mettre d'accord sur le séquençement proposé, car il pourrait y avoir d'autres solutions.

- 1) Si pas de contact, on va tout droit
- 2) Si contact à droite, on inverse la vitesse pour DureeRecul=50 (50x20=1s) en tournant à droite.
- 3) Idem à gauche.

Cela fait 3 sous-tâches pour le déplacement (tâche TkDepl), et une tâche pour les capteurs qui ne semble pas nécessaire ; on s'en occupe pendant la tâche d'avance normale seulement.

```

TkDepl0 :  ; On avance en surveillant les capteurs d'obstacles
        TestSkip,BC  PortC :#bObstDroit
        Jump  ODroit
        TestSkip,BC  PortC :#bObstGauche
        Jump  OGauche
        Ret

ODroit :  ; On passe en tâche 1
        -- on modifie la vitesse en tournant à gauche
        Inc    TkDepl
        Ret

OGauche :  ; On passe en tâche 2

```

```

-- on modifie les vitesses en tournant à droite
Inc   TkDepl
Inc   TkDepl
Ret

TkDepl1 :   ; On recule pendant 1 s
--- on décompte le temps
Ret   ; le temps n'est pas écoulé
; on prépare pour reprendre la ligne droite (ou courbe)
Clr   TkDepl
Ret

TkDepl2 :   ; On recule pendant 1s
-- on décompte le temps
Ret   ; le temps n'est pas écoulé
; on prépare pour reprendre la ligne droite (ou courbe)
Clr   TkDepl
Ret

```

A vous de compléter les instructions. Elle n'ont pas été écrites pour vous montrer la façon dont on écrit le programme : on se préoccupe de la structure avant de coder le détail des actions, et simultanément on complète les liste des constantes et des variable. Il ne doit pas y avoir de nombres dans le programme. Toutes les valeurs doivent être déclarées au début pour être facilement changées.

Le programme complet pour une carte BimoPlus se trouve sous xxxx ??

11.3 Exemple 3 – Ascenseur 3 étages

Un ascenseur a 3 étages, soit un moteur, trois poussoirs pour appeler à l'étage (bAppel1 etc) et trois fin de course pour détecter l'arrivée de l'ascenseur à chaque étage (bStop1 etc).

Les entrées sont sur le portC, les 2 sorties de commande de moteur sur le portA.

Définissons 9 tâches

Etage0 : ; arrêté à l'étages 0, surveille appel1 et appel2

Appel01 : ; on a pesé sur le bouton d'appel de l'étage 1 et l'ascenseur monte jusqu'au fin de course 1

Appel02 : ; on a pesé sur le bouton d'appel de l'étage 1 et l'ascenseur monte jusqu'au fin de course 2

Etage1 : ; etc

Ecrivons le début de ce programme, après les initialisations générales, y compris les instructions qui amènent l'ascenseur à l'étage zéro, s'il n'y est pas.

```

Clr      TaskAsc ; L'ascenseur est à l'étage zero au début
Loop :  Call    DoAsc
; rien d'autre à faire ?
        Jump    Loop

DoAsc :  Move    TaskAsc,W
        Add     W,PCL
NoTk0 = 0
        Jump    Etage0
        Jump    Appel01
        Jump    Appel02
NoTk1 = 3
        Jump    Etage1
        Etc

Etage0 : TestSkip,BC  PortC :#bAppel1
        Jump    Ap01
        TestSkip,BC  PortC :#bAppel2
        Jump    Ap02
        Ret     ; On attend
Ap01:   Move    #Monte,W
        Move    W,PortC
        Inc     TaskAsc,W ; continue en Appel01
        Ret

Ap02:   Move    #Monte,W
        Move    W,PortA

```

```

Inc      TaskAsc,W
Inc      TaskAsc,W      ; continue en Appel02
Ret

```

```

Appel01 :      ; on attend le fin de course de l'étage 1
TestSkip,BS PortC :#bStop1
Ret
Move     #Stop,W
Move     W,PortA
Move     NoTk1,W
Move     TkAsc,W
Ret      ; On continue en Etage1

```

```

Appel02 :      ; on attend le fin de course de l'étage 2
TestSkip,BS PortC :#bStop2
Ret
Move     #Stop,W
Move     W,PortA
Move     NoTk2,W
Move     TkAsc,W
Ret      ; on continue en Etage2

```

Etc, c'est vraiment facile à écrire, facile à tester et facile à modifier. Le programme complet s'appelle **T877TaskAsc.asm**

11.4 Résumé

Décomposer un application en tâches qui se déroulent suffisamment souvent est plus simple qu'une gestion par interruption. Les tâches peuvent souvent se tester séparément en écrivant des programmes simples.

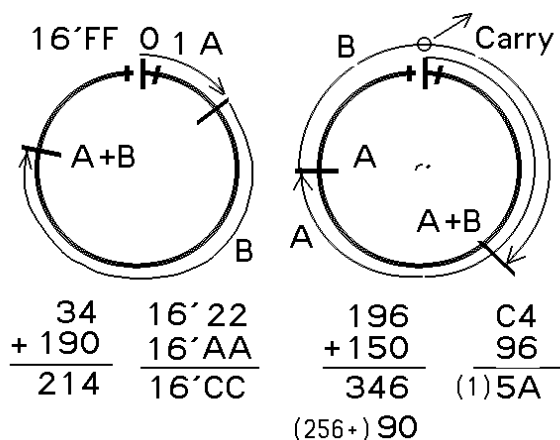
La difficulté est de trouver une période d'échantillonnage assez rapide pour que toutes les tâches soient servies dans le temps voulu, et assez longue pour avoir assez d'instructions pour exécuter chaque sous-tâche.

Les interruptions sont réservées pour des communications rapides, RS232, I2C, et il faut veiller à ce qu'elles soient servies aussi rapidement que possible.

12 Arithmétique

Les PICs ne savent qu'ajouter et soustraire des nombres binaires de 8 bits. S'il y a dépassement de capacité, le bit C (Carry) est mis à un. Si le résultat est nul, le bit Z (Zero) est activé. Une addition peut donner un résultat nul : $128+128=256=0$ (le Carry est activé).

La représentation du cercle arithmétique aide bien à comprendre.



Pour la soustraction, le PIC est différent d'autres processeur puisqu'il ajoute le complément plutôt que de soustraire. L'instruction Sub est aussi inhabituelle, tant mieux si vous n'avez pas d'habitudes, CALM dit exactement ce que font les instructions arithmétiques. Le 1^e opérande est soustrait du 2^e, et le résultat est mis dans le 3^e opérande, s'il y en a un 3^e.

Val est une valeur 8 bits, Reg est une variable ou un registre périphérique

```

Add     #Val,W      ; #Val + W → W      ; pas sur 10F 12F
Add     Reg,W       ; Reg + W → W

```

```

Add   W,Reg      ; W + Reg → Reg
Sub   W,#Val,W   ; #Val - W → W      ; pas sur 10F 12F
Sub   W,Reg,W    ; Reg - W → W
Sub   W,Var      ; Var - W → Var

```

Evitez de soustraire, additionnez le complément ! Add # -5,W soustrait 5 de W et ne donne pas le même résultat que Sub W,#5,W qui soustrait W de -5 (bizarrerie du PIC).

Pour un registre aussi, pour soustraire 5 de Reg, c'est en général préférable d'écrire

```

Move  #-5,W
Add   W,Reg

```

12.1 Complément à 1 et à 2

Le complément à 1 inverse tous les bits, c'est ce que fait l'instruction Not. On utilise cette instruction par exemple si des interrupteurs branchés sur un port sont actifs à zéro et que l'on voudrait convertir les zéros en uns.

Le complément à 2 est le résultat de l'opération zéro moins le nombre. C'est aussi le complément à 1 plus 1. L'arithmétique des nombres négatifs est délicate, des explications, des routines et des macros sont donnés sous www.didel.com/pic/Arith.pdf

12.2 Comparaison

On a souvent besoin de comparer deux nombres positifs et on utilise la soustraction pour cela.

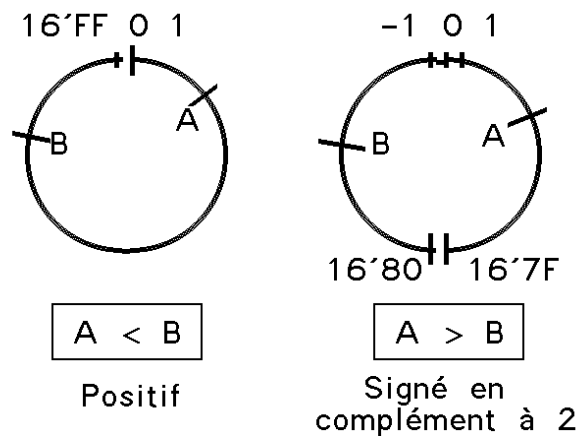
```

Move  Nb1,W
Sub   W,Nb2,W

```

Si C est à 1, Nb1 > Nb2

Si Z est à 1, Nb1 = Nb2



La soustraction de deux nombres positifs peut donner un résultat négatif, représenté en complément à 2. Pour avoir sa valeur absolue, positive, on doit soustraire le résultat de zéro (prendre son complément à 2)

```

Clr   Temp ; variable temporaire
Sub   W,Temp,W ; W → 0-W = -W ou -W → W

```

12.3 Saturation

Souvent, un compteur doit avoir une butée et cela s'implémente avec quelques instructions.

Une addition peut aussi devoir être saturée ; c'est un peu plus délicat puisque le résultat d'une addition peut être petit, s'il y a dépassement de capacité. Avant d'accéder dans une table, il peut être important de saturer la valeur pour éviter de lire en dehors des valeurs prévues.

Le document www.didel.com/pic/Satu.pdf traite les cas les plus fréquents.

12.4 Multiprécision

Ne parlons pas de la multiprécision et des nombres négatifs difficiles à maîtriser. Les PICs ne sont pas fait pour calculer, mais pour manipuler des bits. Toutefois, on doit parfois comparer, modifier des nombres, et plus d'information se trouve sous xxxx.

Des routines et macros très utiles sont décrites sous www.didel.com/pic/Arith.pdf

12.4 BCD (Décimal codé binaire)

Le code BCD représente 2 chiffres décimaux (0 à 9) sur les deux moitiés d'un octet. On peut additionner, soustraire, des nombres BCD, des exemples de routines sont donnés sous xxxx ??.

Compter et décompter en BCD est souvent utile, si le résultat doit être présenté sur un affichage 7 segments. Convertir de binaire en décimal est souvent nécessaire si on interagit avec un clavier écran. Voir www.didel.com/picq/doc/DopiBinD.pdf

12.5 Moyennes

Calculer la moyenne de nombres consécutifs, par exemple pour filtrer un capteur, n'est pas évidente. Le document www.didel.com/pic/Moyennes.pdf montre comment faire.

12.6 Résumé

Pour mettre au point des routines arithmétiques, il est recommandé de faire des programmes de test qui interagissent avec un PC via une liaison série. Le PC transmet le code ASCII des chiffres et signes, qu'il est facile de convertir. Le microcontrôleur renvoie les résultats. Il faut toujours tester soigneusement les opérations particulières (résultat nul, dépassements de capacité, etc.). Le document xxx ?? donne plusieurs exemples <http://www.didel.com/picg/doc/AideDebug.html>

13 Macros et .If

Les macros sont une belle invention pour remplacer des définitions et séquences d'instruction par une seule ligne qui peut avoir des paramètres. Pour les détails, consulter la page 47 de www.didel.com/picg/picg87x/Picg78.pdf
Ce document détaille aussi les pseudos, .Loc, .End si vous voulez en savoir plus.

13.1 Actions sur des périphériques

Il faut définir pour chaque périphérique des macros qui expriment la fonction, et qui codent la réalisation. Par exemple, faire clignoter une Led à l'enclenchement est pratique, la routine qui fait clignoter trois fois est identique pour tous les processeurs, mais elle n'agit pas toujours sur le même bit du même port. Devoir mettre à jour l'instruction qui allume et éteint la Led est source d'erreurs. Il faut déclarer dans le fichier **xxDef.asi** de définition des périphériques:

```
.Macro LedOn
    Set    PortC:#bLed
.Endmacro
.Macro LedOff
    Clr    PortB:#bLed
.Endmacro
```

Ceci permet par la suite d'écrire simplement LedOn LedOff dans les routines que l'on passe d'une application à l'autre sans devoir retoucher le port et le bit.

13.2 Assemblages conditionnels - .If

Les Microdudes ont été développés pour faciliter le développement d'application pour des petits processeurs, mis au point sur un processeur qui a des ports supplémentaires pour afficher des compteurs, variables d'état, impulsions à observer à l'oscilloscope.

Le choix du processeur se fait en déclarant au début du programme principal

```
T877 = 1
T630 = 0
```

C'est de préférence dans un fichier séparé **xxSet.asi** que l'on définit ce qui caractérise le processeur

```
.If    T877                .If    T630
\b;16F877                \b;16F676/630
.Proc  16F877              .Proc  16F630
.Ref   16F877              .Ref   16F630
\b;Configuration         \b;Configuration
.Loc   16'2007             .Loc   16'2007
.L16   16'3F39             .L16   16'3F94
.Endif                    .Endif
```

Dans le fichier **xx.Init.asi** on met les initialisations spécifiques à chaque processeur en séparant par des .If ce qui n'est pas commun.

On a ensuite des fichiers spécifiques pour s'occuper des capteurs, des moteurs, etc. qui n'ont pas de raison de dépendre du processeur.

Le programme principal, xx03.asm (si on numérote les versions de cette façon) contient essentiellement des commentaires et des insertions de fichiers (.Ins xxyy.asi).

Comme exemple on peut ouvrir – exemple avec BimoPlus

14 Structure et mise en page

Maîtriser la programmation prend du temps, et le problème traditionnel est que l'on a de la peine à reprendre d'anciens programmes pour les améliorer, car leur documentation de structure est incomplète.

14.1 Structuration

La première étape pour une bonne technique de programmation est de bien structurer et commenter les programmes. Un commentaire pour chaque ligne est absurde, une ligne vide est souvent un bon commentaire.

Une décomposition en fichiers ayant chacun une fonction claire permet de récupérer plus facilement des modules de programmes.

14.2 Commentaires

Les commentaires dans les programmes sont essentiels pour pouvoir se relire et se remettre dans le bain après quelques jours ou mois, et naturellement pour permettre à un ami de s'y retrouver. Bien commenter n'est pas mettre un commentaire à chaque ligne, mais un commentaire expliquant le rôle d'un groupe d'instructions. Une ligne vide est un bon commentaire puisqu'elle peut montrer comment regrouper la lecture des instructions.

14.3 Noms des variables

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.

14.4 Spécifications des routines

Les routines doivent pouvoir être utilisées sans que cela soit nécessaire de lire leurs instructions. Le commentaire initial doit dire la fonction réalisée, quels sont les registres et variables qui apportent l'information à traiter, quels sont les registres et variables qui rendent de l'information, quels sont les registres utilisés et modifiés. Par exemple une routine qui compte en décimal et pas en binaire se documente et s'écrit :

```
\rout ;IncBCD incrémente en décimal codé binaire
; On compte en binaire, mais si le comptage passe de 9 à A sur les 4 bits de poids faible,
; on corrige en mettant à zéro et en incrémentant les 4 bits de poids fort.
; Si la valeur sur les poids forts passe de 9 à A, on met à zéro et active le Carry
\n; A registre avec une valeur BCD de 1 à 99 : 00 01 .. 09 10 ... 99
\out; A registre augmenté de 1. C=1 si passage de 99 à 00
\mod; A, B, F
```

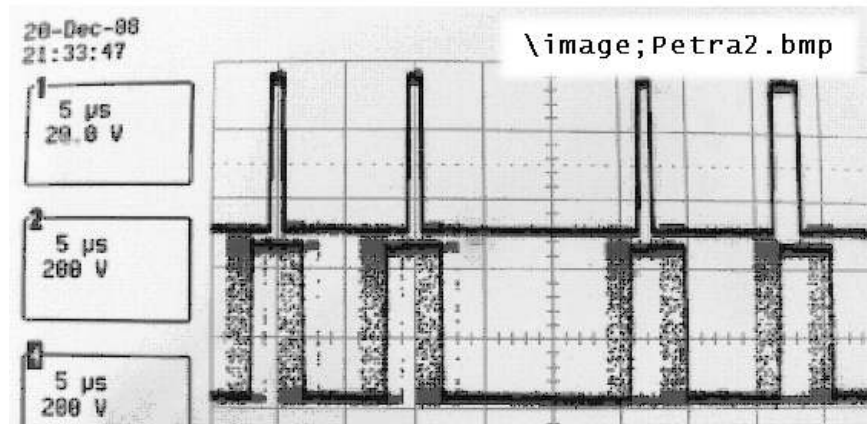
IncBCD :

14.5 Mise en évidence "boa"

Les ordres boa (car ils commencent par une Barre Oblique Arrière) et l'insertion de fichier image (.bmp), avec l'impression multi-colonne de SmileNG est un avantage considérable pour les grands programmes.

On peut se limiter à utiliser les séquences suivantes :

Program	xxxx	yyyy	<code>\prog;xxxx yyyy</code>
Variables	xxxx	yyyy	<code>\var;xxxx yyyy</code>
Macro	xxx	yyyy	<code>\macro;xxx yyyy</code>
Table	xxxx	yyyy	<code>\table;xxxx yyyy</code>
Routine	xxxx	YYYY	<code>\rout;xxxx YYYY</code>
In:	xxxx		<code>\in;xxxx</code>
Out:	xxxx		<code>\out;xxxx</code>
Mod:	xxxx		<code>\mod:xxxx</code>
xxxx			<code>\b;xxxx</code>



On remarque qu'il est possible d'insérer des fichiers images en .bmp (prennent malheureusement beaucoup de place. Ceci aide considérablement la lisibilité des programmes, puisque l'on peut insérer un schéma, un organigramme, le timing d'un oscilloscope, une photo d'un testeur.

SmileNG permet de définir dans son menu Tools-Editor options la couleur et la taille pour les définitions, instructions, commentaires. La touche F8 commute entre le mode qui interprète les séquence boa et le mode qui les affiche en clair.

15 Dépannages et compléments

15.1 Déverminage

La mise au point des programmes n'est pas toujours facile car on ne voit pas ce que fait le processeur. Des simulateurs existent, comme avec le Dauphin, mais le problème est en général à la 1000^e instructions, et si on peut mettre un point d'arrêt pour observer l'état, on ne peut pas bloquer les signaux entrants, une télécommande infrarouge par exemple. Avec les processeurs simples, il n'y a pas d'autre solution que de laisser tourner le contrôleur, mais on lui demande de créer des impulsions, afficher des valeurs sur des ports non utilisés par l'application. C'est l'intérêt du 16F877 pour mettre au point les programmes pour des processeurs plus simples. Différentes techniques sont expliquées dans www.didel.com/picq/doc/DopicBug.pdf

15.2 Sottisier

Oui, vous ferez des bêtises de programmation, et des grosses ! Certaines font perdre beaucoup de temps. Un document liste les plus fréquentes www.didel.com/pic/Sottisier.pdf

15.3 Compatibilité 12 bits

La famille PIC a des processeurs en boîtier 6, 8 et 14 broches qui ont des instructions 12 bits. Trois instructions sont supprimées, et les adresses sont limitées. Avec quelques macros et .lf, on développe sur 877 des applications pour 10F 12F et transfère le programme testé sur la petite puce 10F 12F au dernier moment. Voir <http://www.didel.com/picq/doc/DopiComp.pdf>

15.4 Optimisation de code

Les PICs permettent parfois des solutions très élégantes auxquelles on ne pense pas toujours. La condition ET sur 2 bits est facilement programmée dans un PIC. Par exemple, si on veut savoir que les bits 3 et 6 de la variable Cz sont à 1, on écrit

```
TestSkip,BC Cz:#3
TestSkip,BS Cz:#6
Jump La condition ET n'est pas réalisée
On continue ici si la condition ET est réalisée
```

Ceci s'applique bien à un compteur par 10 (ou 6, 12, etc), puisqu'il faut mettre le compteur à zéro l'état limite est atteint. Si le compteur par 10 est programmé sur le portB (dont seuls les 4

bits de poids faibles sont utilisés), on peut écrire la routine qui incrémente le portB de la façon suivante:

```
IncBCD: Inc   PortB
        TestSkip,BC   PortB:#1
        TestSkip,BS   PortB:#3
        Ret
        Clr   PortB
        Ret
```

Le fichier

[Arith](#) autres ex

15.5 Retards courts

On a vu comment faire des retards longs. Lorsque l'on doit générer des impulsions calibrées, par exemple du 38 kHz pour de l'infrarouge, on doit ajuster des retards à la microseconde près, en utilisant un nombre minime d'instructions.

Le retard le plus court est celui de l'instruction Nop, 1 us à 4 MHz. L'instruction Jump APC+1 prend 2 us. Si la routine Rien : Ret existe dans un coin, l'instruction Call Rien dure 4 us. Des boucles d'attente sont ensuite utilisées.

15.6 Watchdog et bits de configuration

Le programme peut dérailler suite à un parasite, ou une condition exceptionnelle non prévue. L'idée pour se protéger est un compteur que le programme remet de temps en temps à zéro avant qu'il arrive à la fin de son comptage et force le redémarrage du programme. Cette option ne doit pas être activée/désactivée par une instruction. Un bit dans le mot de configurations autorise ce fonctionnement. Le fichier www.didel.com/pic/Wdog.pdf explique en détails et documente quelques bits du mot de configuration.

16 Programmes de test

La documentation d'un microcontrôleur prend des centaines de pages, et les différences d'un microcontrôleur à l'autre ne sont jamais documentées.

Si l'on veut programmer du PWM, des transferts synchrones, le mieux est de faire un programme de test qui permet de vérifier le bon fonctionnement avant d'intégrer dans l'application. Un oscilloscope est souvent nécessaire pour cette application.

Un certain nombre de programmes de test sont documentés pour 6 Pics différents sous www.didel.com/pic/PicTests.pdf

17 Routines et programmes exemple

Routines non adaptées, Testées en 1998-2000

<http://www.didel.com/picg/piclib/DopicSources.html>

Idées de programme

- Horloges
- Compteur de réflexe
- Dé électronique affiché sur l'écran
- PacMan
- MasterMind
- Puissance 4
- Nim (jeu des allumettes)
- Jeu de la vie

PIC 16F877 - CALM instructions

www.didel.com/pic/CalmInstr877.pdf

Code 14 bits	Microchip	CALM	Flags	Operation
110000 kkkkkkkk	movlw val	Move	#Val,W	[none] #Val → W
000000 1fffffff	movwf reg	Move	W,Reg	[none] W → Reg
001000 0fffffff	movf reg,0	Move	Reg,W	[Z] Reg → W
001000 1fffffff	movf reg,1	Test	Reg	[Z] Reg → Reg
000000 01100010	option	Move	W,Option	[none]
000000 01100101	tris porta	Move	W,TrisA ¹	[none]
000000 01100110	tris portb	Move	W,TrisB ¹	[none]
000000 01100111	tris portc	Move	W,TrisC ¹	[none]
111110 kkkkkkkk	addlw val	Add	#Val,W	[C,D,Z] #Val+W → W
000111 0fffffff	addwf reg,0	Add	Reg,W	[C,D,Z] Reg+W → W
000111 1fffffff	addwf reg,1	Add	W,Reg	[C,D,Z]
111100 kkkkkkkk	sublw val	Sub	W,#Val,W	[C ² ,D,Z] #Val+(-W) → W
000010 0fffffff	subwf reg,0	Sub	W,Reg,W	[C ² ,D,Z] Reg+(-W) → W
000010 1fffffff	subwf reg,1	Sub	W,Reg	[C ² ,D,Z] Reg+(-W) → Reg
111001 kkkkkkkk	andlw val	And	#Val,W	[Z]
000101 0fffffff	andwf reg,0	And	Reg,W	[Z]
000101 1fffffff	andwf reg,1	And	W,Reg	[Z]
111000 kkkkkkkk	iorlw val	Or	#Val,W	[Z]
000100 0fffffff	iorwf reg,0	Or	Reg,W	[Z]
000100 1fffffff	iorwf reg,1	Or	W,Reg	[Z]
111010 kkkkkkkk	xorlw val	Xor	#Val,W	[Z]
000110 0fffffff	xorwf reg,0	Xor	Reg,W	[Z]
000110 1fffffff	xorwf reg,1	Xor	W,Reg	[Z]
001110 0fffffff	swapf reg,0	Swap	Reg,W	[none] Reg → Reg swapped Reg → W
001110 1fffffff	swapf reg,1	Swap	Reg	[none] Reg → swapped Reg
000001 1fffffff	clrf reg	Clr	Reg	[Z=1] 0 → Reg
000001 00000000	clrw	Clr	W	[none] 0 → W
000000 01100100	clrwtd	ClrWDT		[none]
000000 01100011	sleep	Sleep		[none]
001001 1fffffff	comf reg,1	Not	Reg	[Z] Reg → .not.Reg
001001 0fffffff	comf reg,0	Not	Reg,W	[Z] Reg → Reg .not.Reg → W
001010 1fffffff	incf reg,1	Inc	Reg	[Z] Reg+1 → Reg
001010 0fffffff	incf reg,0	Inc	Reg,W	[Z] Reg → Reg Reg+1 → W
000011 1fffffff	decf reg,1	Dec	Reg	[Z] Reg-1 → Reg
000011 0fffffff	decf reg,0	Dec	Reg,W	[Z] Reg → Reg Reg-1 → W
001101 1fffffff	r1f reg,1	RLC	Reg	[C] Reg → .rlc.Reg
001101 0fffffff	r1f reg,0	RLC	Reg,W	[C] Reg → Reg .rlc.Reg → W
001100 1fffffff	rrf reg,1	RRC	Reg	[C] Reg → .rrc.Reg
001100 0fffffff	rrf reg,0	RRC	Reg,W	[C] Reg → Reg .rrc.Reg → W
0100bb bfffffff	bcf reg,b	Clr	Reg:#b	[none]
0101bb bfffffff	bsf reg,b	Set	Reg:#b	[none]
001111 1fffffff	incfsz reg,1	IncSkip,EQ	Reg	[none] Reg+1 → Reg
001111 0fffffff	incfsz reg,0	IncSkip,EQ	Reg,W	[none] Reg → Reg Reg+1 → W
001011 1fffffff	decfsz reg,1	DecSkip,EQ	Reg	[none] Reg-1 → Reg
001011 0fffffff	decfsz reg,0	DecSkip,EQ	Reg,W	[none] Reg → Reg Reg-1 → W
0110bb bfffffff	btfsz reg,b	TestSkip,BC	Reg:#b	[none]
0111bb bfffffff	btfss reg,d	TestSkip,BS	Reg:#b	[none]
1010kk kkkkkkkk	goto addr	Jump	Addr	[none]
1000kk kkkkkkkk	call addr	Call	Addr	[none]
110100 kkkkkkkk	retlw val	RetMove	#Val,W	[none]
000000 00001000	return	Ret		[none]
000000 00001001	retfie	Retl		[int]
000000 00000000	nop	Nop		[none]

bcf status,0	CLRC	Clr F:#0
bsf status,0	SETC	Set F:#0
btfsz status,0	Skip,CS	TestSkip,BS Status:#C
btfsc status,0	Skip,CC	TestSkip,BC Status:#C
btfsz status,2	Skip,EQ	TestSkip,BS Status:#Z
btfsc status,2	Skip,NE	TestSkip,BC Status:#Z

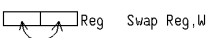
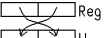
F same name as Status

Note 1: Move W,TrisD, Move W,TrisE does not exist.

Use Move W,PortD, Move W.PortE in Bank1

Note 2: Processor add the 2-s complement; Carry value is inverted compared to most other µP

Meaning of non trivial instructions

Swap Reg  Swap Reg,W 

IncSkip,EQ Reg Increment Reg and skip next instruction if result is equal to zero

IncSkip,EQ Reg,W Copy Reg into W, increment W and skip next instruction if result is equal to zero

TestSkip,BC Reg:#b Test bit number b in Reg and skip if this bit is clear (zero)

,BS ... if this bit is set (one)

RetMove #Val,W Return from routine with immediate value Val prepared into W

Move #Reg,W }
Move W,FSR } ≡ Move Reg,W Indirect access to registers.
Move {FSR},W } FSR can be incremented, etc.

Clrc Clear Carry bit

Setc Set Carry bit

Skip,CS Skip next instruction if Carry bit is set (one) pic84in

All instructions execute in 4 clock cycles (1 µs at 4 MHz) except the Skip, Jump, Call and Ret that takes the double
Compare for equality: Xor W,Reg or Xor Reg,W Compare for Inequality: Sub instruction, but be careful
Except I/O, 16F87x, and other 14-bit instruction PICs have the same instruction set. jdn090823