Architecture des PICs

Ce document présente l'architecture des microcontrôleurs PICs pour un lecteur qui n'a pas de connaissances particulières, sans rentrer dans trop de détail. Il complète les cours Kidule, 628, 877 et d'autres documents qui montrent comment programmer et insistent plus sur le répertoire d'instructions.

Un premier modèle simplifié du PIC

Un microcontrôleur a comme première mission de lire des entrées, et décider les valeurs à assigner aux sorties.
Les PICs 16Fxxx peuvent être vus en première approche comme comportant des ports d'entrée-sortie plus ou moins complets et spécialisés, avec leurs registres de direction associés. Les port A, B, C sont aux adresses 5, 6, 7 (l'assembleur le sait, il n'y a pas besoin de le lui dire). Les registres de direction TrisA, TrisB et TrisC définissent si le bit du port est en entrée ou en sortie, comme on verra plus loin.

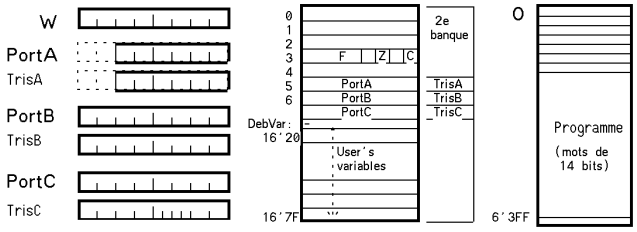

Fig 1 Modèle pour les ports et les 2 mémoires du PIC 16F870

Une mémoire en écriture et lecture prolonge la zone des entrées-sorties et registres de contrôle et stocke les variables (compteurs, registres de calcul, valeurs temporaires).
Les instruction sont dans une mémoire programme, en lecture seulement (l'écriture se fait avec un programmateur à partir du programme traduit en binaire). Elles sont exécutées l'une après l'autre en général et analysées par le décodeur d'instructions.
Un registre W joue un rôle particulier et fait penser aux plaques tournantes dans les anciennes gares de locomotives. La loco est une valeur qui passe par W pour aller dans le hangar (ports ou variables), vers la gare (entrées-sorties) ou dans le dépôt (test, modifications).

Lorsque le processeur exécute par exemple l'instruction "Move W,PortA", une impulsion charge le registre appelé PortA ou RA (register A). Les sorties qui ont été préparées en sortie copient cette information. W --> registre interne --> sorties. Si on relit avec un "Move PortA,W", on lit directement l'état des broches.

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

```
Move   #Valeur,W      ; Le nombre Valeur est copié dans W
Move   W,MaVariable   ; Le contenu de W est copié dans MaVariable
Clr    Compteur       ; Initilalise la variable compteur avec la valeur zéro (2'00000000)
Add    W,Total        ; Additionne le contenu de W au contenu de Total, et met le résultat dans Total
```

Le registre F ou STATUS (flags, état) à l'adresse 3 mémorise en particulier deux bits que l'on trouve dans tous les processeurs: C est le "Carry", activé par une addition avec dépassement de capacité, ou par un décalage. Z est le "zéro bit", activé si le résultat d'une opération est nul (les 8 bits du résultat transféré dans W ou dans un registre sont nuls). Toutes les instructions n'agissent pas sur Z et sur C: la feuille de codage précise pour chaque intruction les bits modifiés.

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

Un programme commence toujours par une initialisation des registres de direction des ports utilisés. Si on ne fait rien, les ports sont en entrée après un Reset. Il faut aussi initialiser les variables, compteurs dont la valeur à l'enclenchement est importante.
On donne un nom aux variables, en listant dans les programmes simples leur emplacement mémoire.

Var1= 16'20 (mieux: DebVar+0),
Var2 = 16'21 (DebVar+1)

Un programme doit spécifier le processeur, pour que l'assembleur vérifie que les instructions sont compatibles. Il définit ensuite les noms constantes et les variables ; le programme en mémoire ne contient que des nombres, mais c'est illisible pour nous.

Le programme est structuré en modules, routines, tableaux, avec des commentaires de structure. La mise en page facilite la lecture, et SmileNG est fier de ses séquences ''boa''.

Le programme se termine par un .End qui dit à l'assembleur que son travail de traduction est terminé. Les lignes suivantes sont ignorées. Cela peut être le mode d'emploi de votre programme. Documentez le but et la structure autant que possible dans votre programme et pas ailleurs. Après le .End, il n'y a plus besoin de ; ou \ devant les lignes, et les lignes peuvent avoir plus de 96 caractères.

Un peu plus sur l'architecture du PIC

Les bits Z, C et D, réunis dans un registre de "fanions" (flag register F ou STATUS) sont très importants et nous donneront un peu de fil à retordre. Le bit Z est activé par quelques instructions pour signaler que le résultat de l'opération est nul. Par exemple l'instruction "XOR Var,W" calcule le ou exclusif de la variable Var et du contenu du registre W. Z est activé si le résultat est nul, ce qui veut dire que Var et W avaient le même contenu (après exécution de l'instruction la variable Var est inchangée et W vaut zéro dans ce cas). "Move W,Var" ne modifie pas Z, mais "Move Var,W" modifie Z (Z=1 si Var contient zéro). Test Var est en fait un Move Var,Var, et modifie Z. Mais Test TMR0 ne doit pas être utilisé pour savoir si TMR0 est à zéro !
Il faut, au début, regarder chaque fois la feuille de codage (voir plus loin) pour vérifier ce que fait exactement l'instruction.
Le bit C est activé s'il y a dépassement de capacité dans une addition, soustraction ou décalage.
Le bit D n'est activé que par l'addition et la soustraction; il est utile pour calculer en décimal, mais il est rarement utilisé.

Banques
Avec ses instructions de 14 bits, les PICs n'ont pas assez de bits pour sélectionner assez de registres et variables. On prend les bits qui manquent dans le registre Status. C'est comme les pages d'un livre. En page 0 (Bank0) on a les ports (jusqu'à l'adresse 20) et en général toutes les variables de l'utilisateur. En Bank1 on a les registres Tris et d'autres registres pour périphériqes. Si on change de banque, il ne faut pas oublier de revenir ! Si on veut lire en Eeprom par exemple, il faut accéder des registres qui sont dans deux banques différentes, les banques 3 et 2 (avec le 16F877).

```
Move  #3,W
Move  W,Status
Clr   EECON1:#EEPGD     ; En banque 3
Set   EECON1:#RD
Move  #2,W
Move  W,Status
Move  EeData,W    ; en banque 2
Move  #0,W
Move  W,Status    ; banque 0 pour la suite
```

Ceci n'est pas une bonne façon d'écrire, poisque tous les autres bits du registre Status sont mis à zéro. On utilise les instructions Set et Clr bits,
Les macros définies dans les fichies xxSet.asi ont la forme

```
.Macro      Bank0to3
  Set Status :#RB0
  Set Status :#RB1
.Endmacro
```

Ces commutations de banque embêtent dans l'initialisation des périphérique, et les programmes PicTest sont là pour donner des exemples d'initialisation. Dans le programme de l'utilisateur, on change rarement de banque et pour des interventions très spéciales.

Registre de configuration
Prenons l'exemple de l'oscillateur. Un client veut mettre un quartz externe, un autre se contente de la précision d'un oscillateur interne.
Le fabricant ne vas pas faire deux microcontôleurs différents. Un bit dans une mémoire spéciale va choisir le mode. Ce bit est dans un registre de configuration, à l'adresse 2007 en général.
Ceci explique les lignes qui terminent un programme (elles peuvent aussi se mettre avant le programme) :

```
.Loc  16'2007
.16   16'3F39
```

Registre de direction Tris
Pour les spécialistes !
On a dit que les registres de directions sont en banque 1. Si le PortA est à l'adresse 5, TrisA est à l'adresse 16'85, mais c'est une adresse que l'assembleur n'accepte pas, puisqu'il n'y a que 7 lignes d'adresse (maximum 16'7F).
TrisA est donc déclaré = 5 et l'instruction Move W,TrisA doit être précédée de Set Status:RB0 pour que l'accès se fasse dans la bonne banque.
Si on écrit Move W,PortA, le résultat est le même puisque PortA = 5.
Historiquement, trois instructions ont été définies pour transférer directement W dans le registre de direction A, B, C, quelles que soit la banque sélectionnée. Ces instructions n'existent pas pour les ports D, E, etc.
Le problème est que Calm a noté ces instructions Move W, TrisA Move TrisB Move W,TrisC, et accepte actuellement les instructions Move W,TrisD Move W,TrisE qui n'ont pas d'effet.
Dans un futur proche, Move W,TrisD Move W,TrisE (pour les processeurs à 40 pattes) devront s'exécuter avec la banque 1 sélectionnée.
En attendant, l'initialisation des ports peut s'écrire comme suit :

```
Set   Status:#RP0   ; ou macro Bank1
Move  #0,W          ; sorties
Move  W,TrisA       ; même effet dans toutes les banques
Move  W,TrisB
Move  W,TrisC
Move  W,PortD ;TrisD
Move  W,PortE ;TrisE
Clr   Status:#RP0   ; ou macro Bank0
```

Pile des adresses de retour
Les PICs ont une pile de 8 positions (2 positions pour les 10F 12F). Si on déborde, la plantée est assurée. Le processeur continue à exécuter des instructions, et peut retomber sur votre programme !
On ne peut pas initialiser/réinitialiser cette pile. Cela permet de faire des choses interdites dans d'autres processeurs, comme mettre une sortie d'erreur dans une routine avec un Jump.

Limitation mémoire
Les adresses des PIC de la famille 16F ont 12 bits, ce qui limite à 14k de mémoire. Le 16F877 en a plus, difficile à exploiter, voir www.didel.com/pic/MultiPages.pdf

Instructions et structuration
Notre meilleur texte (le plus récent) utilise le kit Eval2840/877 http://www.didel.com/pic/Cours877.pdf et offre des exercice progressifs et des exemples qui permettent de créer ses propres applications avec des PICs de toute taille.

Programmer dans n'importe quel langage nécessite une bonne compréhension de l'environnement de programmation, qui dépends de l'application.
Programmer en C est dit plus facile car les initialisations, commutations de banque, etc, sont cachées et les applications choisies demandent en général de calculer et afficher des textes, ce que l'on apprend à l'école.
Gérer des bits et des périphériques n'est pas plus facile en C qu'en assembleur. Alors courage !

jdn 090820/090922/100926