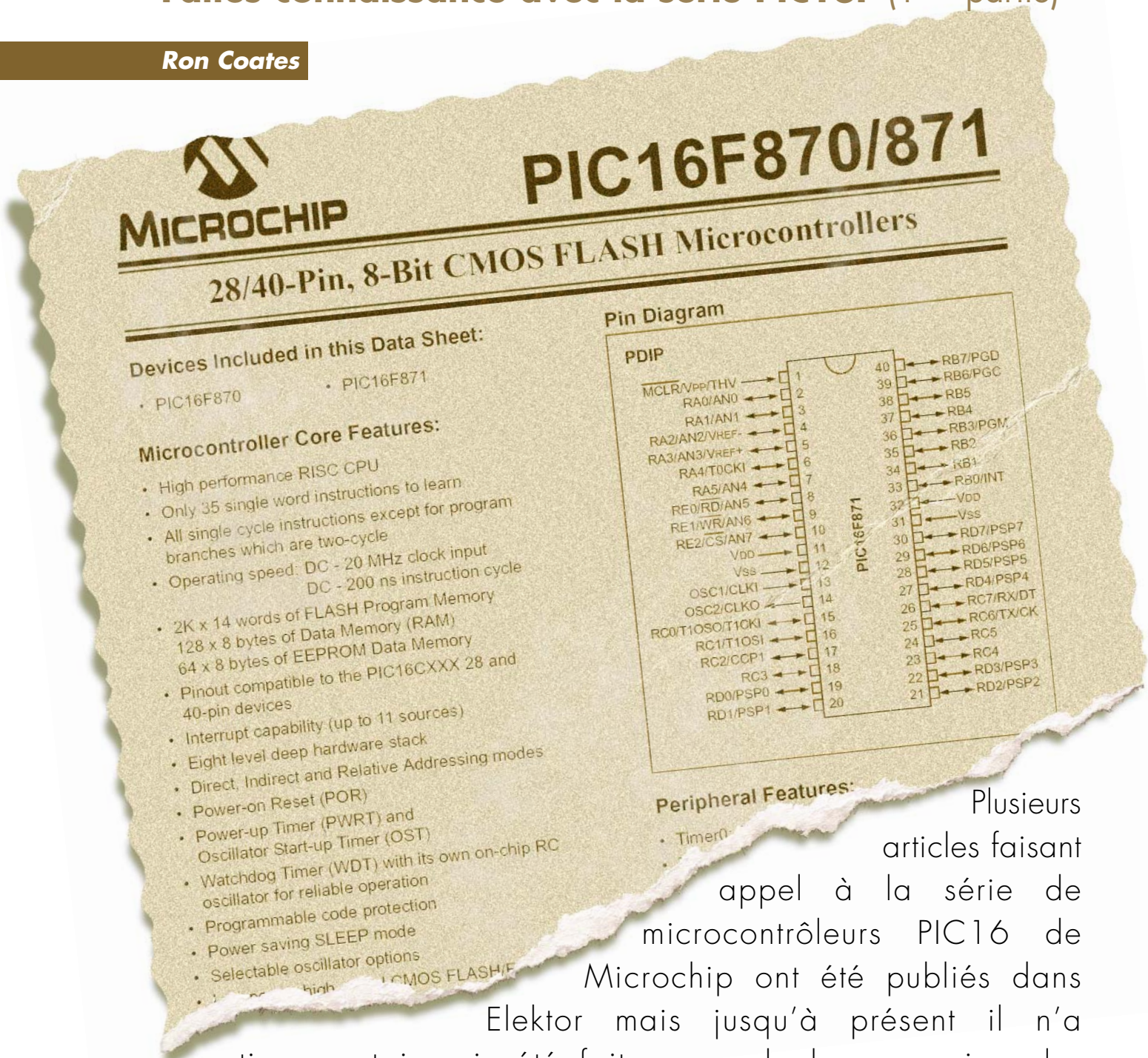
ADIEU «16», BIE

Faites connaissance avec la série PIC18F (1ère partie)

Ron Coates



Plusieurs articles faisant appel à la série de microcontrôleurs PIC16 de Microchip ont été publiés dans Elektor mais jusqu'à présent il n'a pratiquement jamais été fait usage de leurs cousins plus puissants de la série 18F. Les microcontrôleurs de la série 18Fxx2 récemment commercialisée sont compatibles broche à broche avec leurs prédécesseurs de la série 16F87x et contiennent quelques notables améliorations.

NVENUE PIC18F

Dans ce premier article d'une série de deux, nous examinerons brièvement les principales différences entre les séries 16F et 18F et en détail les changements au niveau de la structure de la RAM (mémoire de données). Le **tableau 1** en résume les principales caractéristiques par rapport à la série 16F.
Il y a une augmentation en ROM et une grosse augmentation en RAM et, pour certaines personnes, rien que cela justifiera leur utilisation. Mais il y a également plein d'autres améliorations :

– Horloge jusqu'à 40 MHz (au lieu de 20).
– Multiplicateur câblé à cycle unique.
– Grandes tables de recherche.
– Capacité à écrire en ROM Flash pendant l'exécution de programmes.
– De nombreuses nouvelles instructions.
– Adressage très nettement amélioré, à la fois pour la RAM et la ROM.

La mémoire et ses accès

Parmi les principales améliorations par rapport à l'ancienne série PIC, il n'y a pas que la capacité augmentée en RAM et en ROM mais également les manières de l'adresser. Avant de discuter ce point en détail, il faut clarifier certains termes que Microchip utilise.

EEPROM dont la signification est immédiate dans la mesure il n'est jamais fait référence à rien d'autre que de l'EEPROM. L'EEPROM est non-volatile et son contenu peut être modifié, non seulement durant la programmation du circuit, mais aussi par le code exécuté par le PIC. L'EEPROM est également l'espace de mémoire qui est le plus employé pour l'enregistrement des préférences de l'utilisateur. Par exemple, l'affichage de la température d'un système de mesure qui pourrait se faire en degrés centigrades ou Fahrenheit. Si la variable de contrôle de cette conversion est stockée en EEPROM, l'utilisateur peut la changer et sa valeur persistera même en l'absence de tension d'alimentation.

RAM et Flash ROM sont respectivement référencées par Microchip comme mémoire de données et mémoire de programme. Source de confusion certaine, spécialement avec la série PIC18F qui permet le stockage en ROM de grands blocs de données matérialisant des tables de recherche. Les données stockées en RAM sont perdues quand la tension d'alimentation du PIC disparaît. Par contre, les données présentes en ROM seront préservées. Tant la RAM que la ROM peuvent être modifiés par le programme en cours d'exécution. Dans la première application, l'auteur a utilisé un circuit de la série 18 dont 12 K de ROM servaient aux données et moins d'1 K pour le programme. Dans cet article en deux parties, nous utiliserons les termes RAM et ROM pour référencer ces zones de mémoire.
Le **tableau 2** donne la cartographie de la mémoire RAM. Pour adresser la RAM directement, on ne dispose que d'un octet (« octet d'adressage ») pour référencer une position (adresse). Typiquement, on donne alors aux zones de mémoire –habituellement appelées registres– un nom en rapport avec leur signification et on les définit au début du programme, par exemple :

```
COUNT EQU    0x24
```

La valeur 0x24 est l'emplacement absolu (en hexadécimal) dans la RAM et vous ne disposez que d'une série de 256 adresses possibles. Comme il y a largement plus de 256 octets de RAM, un système de « banc » doit être utilisé afin de pouvoir toutes les accéder. C'était également le cas avec la série PIC16 mais l'arrangement de la série PIC18 est complètement différent.
Il y a potentiellement seize bancs disponibles, numérotés de 0 à 15. Seuls les bancs 0, 1, 2 et 15 sont implémentés dans les 242 et 442. Les 252 et 452 disposent en plus des bancs 3,4 et 5. Le banc 15 est monopolisé par les Registres de Fonctions Spéciales (SFR pour *Special Function Register*), de sorte que seulement trois ou six bancs (768 ou 1 536 octets) sont disponibles pour des registres à usage général. Ces bancs peuvent être sélectionnés grâce au Registre de sélection de Banc BSR (*Banc Select Register*) mais, comme nous allons le voir, nous n'aurons normalement pas à nous en soucier.

Adressage Direct

L'adressage direct est beaucoup plus facile qu'avec la série PIC16F et nous avons de sérieux doutes à propos de l'utilisation que feront en finale la plupart des programmes des commandes relatives aux bancs.
Comme vous pouvez le constater à partir de la cartographie de la mémoire, la première moitié du banc 0 et la seconde moitié du banc 15 ont été dénommées Banc d'Accès *(Access Bank*). La lecture des fiches techniques de Microchip pourrait vous donner à penser que cela vous donnerait 128 registres à usage général et tous les Registres de Fonctions Spéciales auxquels vous pourriez accéder directement sans utiliser les commandes relatives aux bancs.
Rien que cela aurait déjà été bien mais en réalité c'est encore mieux puisque vous pouvez en pratique utiliser tous les 256 registres du banc 0 et tous les Registres de Fonctions Spéciales sans faire appel aux commandes de gestion des bancs et sans risques de conflits d'adressage. Puisque nous avons dit peu avant que l'on ne pouvait jamais accéder qu'à 256 octets sans changer de banc, comment cela se peut-il ?
Cela se fait en utilisant le bit d'accès (a) pour choisir soit le banc d'accès, soit le banc sélectionné par le BSR. Supposons que l'on définisse un registre nommé TESTREG comme suit :

```
TESTREG        EQU     0x84
```

Un coup d'œil à la cartographie du Registre de Fonctions Spéciales vous montrera que 84 dans le Banc d'Accès correspond au Port E. Vous pourriez penser que cela pourrait engendrer un conflit entre ces deux zones et cela se peut en effet mais c'est pour l'éviter que Microchip a défini certaines valeurs par défaut pour le bit d'accès.

Tableau 1. PIC16 vs PIC18F – comparaison rapide

Code de type	Boîtier	E/S (lignes)	RAM (octets)	ROM FLASH (octets)	EEPROM (octets)
16F873	DIL 28	22	192	4 k	128
16F874	DIL 40	33	192	4 k	128
16F876	DIL 28	22	368	8 k	256
16F877	DIL 40	33	368	8 k	256
18F242	DIL 28	22/23 *	768	8/16 k **	256
18F442	DIL 40	33/34 *	768	8/16 k **	256
18F252	DIL 28	22/23 *	1536	16/32 k **	256
18F452	DIL 40	33/34 *	1536	16/32 k **	256

* La broche de sortie additionnelle (RA6) peut être utilisée à la place de la broche OSC2 en cas d'utilisation d'une horloge externe.
** Cf. description de la ROM Flash pour la raison de ces 2 valeurs.

Considérons le code suivant :

```
MOVLW 02
MOVWF TESTREG

MOVLW 04
MOVWF PORTE
```

Bien que nous n'ayons pas utilisé le bit d'accès, ces lignes seront traduites par le compilateur comme si nous avions écrit :

```
MOVLW 02
MOVWF TESTREG,1

MOVLW 04
MOVWF PORTE,0
```

Dans le cas de la première instruction MOVWF, le bit d'accès vaut 1, ce qui signifie que l'on utilise le banc pointé par le BSR dont la valeur par défaut est zéro. La valeur 02 va donc être stockée à l'adresse 0x084.
Pour la seconde instruction MOVWF, le bit d'accès vaut 0, ce qui signifie que l'on utilise le banc d'accès. La valeur 04 sera donc stockée en 0xF84.

L'astuce est que l'on ne doit pas se préoccuper de cela car le compilateur s'en charge pour nous, en supposant bien sûr que toutes les « déclarations » (étiquettes et/ou déclarations de fonctions) figurent au début du programme.
Ne vous inquiétez pas si vous ne comprenez pas tout à fait ce qui précède, tout ce qui compte c'est que vous pouvez utiliser tous les Registres à Usage Général de 00 à FF et tous les Registres de Fonctions Spéciales dont vous avez besoin sans avoir à vous préoccuper du tout de la gestion des bancs. Au cas où vous auriez besoin de plus de 256 registres, vous devrez sans doute de toute façon utiliser l'adressage indexé.

Tableau 2. Cartographie de la RAM d'un PIC18F

Banc n°/nom	Plage des adresses	Notes
0	000 – 0FF	00 – 7F est Banc d'accès
1	100 – 1FF	
2	200 – 2FF	
3 - 5	300 – 5FF	Si présent
6 -14	600 – EFF	Non utilisé
15	F00 – F7F	Non utilisé
Special Function Registers	F80 – FFF	(Access Bank)

Adressage Indirect

Si vous pensiez que l'adressage direct n'est pas mal du tout, les choses sont encore améliorées par l'adressage indirect. C'est parce que nous disposons d'un nombre à 12 bits pour l'adressage, ce qui nous donne un accès direct à l'ensemble des 16 bancs. Après tout tant 2^{12} que 16 x 256 valent 4 096.
Tout d'abord, il y a maintenant trois registres FSR : FSR0, FSR1 et FSR2. Ils sont associés à trois registres INDF. Pour les besoins de notre explication, nous ne parlerons que de FSR0 et INDF0, mais les mêmes règles s'appliquent aux deux autres paires.
La grosse différence est qu'il y a maintenant un registre de 8 bits FSR0L (L = Low) qui est équivalent du registre FSR de la série 16F. Nous avons également un registre FSR0H (H = High) dont seuls les quatre bits de poids faible sont disponibles puisqu'ils servent à adresser les seize bancs.

En utilisant conjointement ces deux registres, nous pouvons « pointer » n'importe quelle zone de l'espace mémoire RAM. Encore mieux : si nous incrémentons FSR0L, que nous atteignons la valeur 0FF et que nous l'incrémentons encore, FSR0L repasse à 00 et FSR0H est automatiquement incrémenté et donc nous passons progressivement par exemple de l'adresse FF du banc 2 à l'adresse 00 du banc 3.

Et il est possible de faire encore mieux grâce aux autres méthodes de manipulation de l'adressage indirect qui ont été introduites.

Il y a ainsi (*n* pouvant prendre la valeur 0, 1 ou 2) :

`INDFn` Ne fait rien de plus qu'exécuter l'instruction sur base du contenu du registre indirect. Exactement comme pour INDF de la série 16.

Latin (de cuisine ?)

Nous pouvons fort bien nous imaginer que cet article soit délicat à saisir pour un débutant. La pierre d'achoppement la plus grande est le nombre sans cesse croissant d'abréviations utilisées par les fabricants et les programmeurs expérimentés.

PIC	Si a l'origine on a pensé que Microchip voulait baptiser ses produits Programmable Interface Controller, il s'est avéré depuis que cette abréviation ne signifie en fait rien.
RAM	Random Access Memory = mémoire à accès aléatoire
ROM	Read-Only Memory = mémoire à lecture seule
DIL	Dual-In-Line = à 2 rangées
F	pour les codes de type de PIC, signifie normalement « Flash »
EEPROM	Electrically Erasable Programmable Read Only Memory
BSR	Bank Select Register = registre de sélection de banc
REG	Register = registre
SFR	Special Function Register = registre à fonction spéciale
GPR	General Purpose Register = registre à usage général

`POSTDEC`*n* Comme ci-dessus mais FSR est automatiquement décrémenté après l'exécution de l'instruction.

`POSTINC`*n* Après l'exécution de l'instruction, FSR est incrémenté.

`PREINC`*n*`FSR` est incrémenté avant l'exécution de l'instruction.

`PLUSW`*n* la valeur contenue dans le registre W est utilisée en temps qu'offset sur la valeur de FSR. L'exécution de l'instruction ne modifie pas le contenu des deux registres W et FSR.

Exemple d'utilisation de la dernière de ces instructions :

```
INCF   PLUSW0
```

Si les valeurs contenues dans FSR0 et W étaient respectivement 0x023 et 04, alors l'effet de l'instruction ci-dessus serait d'incrémenter la valeur contenue à l'adresse 0x027. Après exécution, les valeurs de FSR et W seront inchangées.
Considérez les morceaux de code suivants que vous pourriez utiliser toujours utiliser dans la partie initialisation de n'importe quel programme pour effacer la RAM.

Pour la série 16F, nous aurions écrit :

```
        MOVLW    0x20
        MOVWF    FSR
NEXT    CLR        INDF
        INCF       FSR
        BTFSS      FSR,7
        GOTO       NEXT
```

Ce morceau de code commence à l'adresse mémoire 0x20 (après les Registres de Fonctions Spéciales) et efface chaque emplacement de mémoire jusqu'à 0x7F, c'est-à-dire tout le banc 1. Comme 0x80 à 0x9F sont les Registres de Fonctions Spéciales, vous auriez dû écrire des morceaux de codes distincts pour les bancs 2, 3 et 4. Pour les composants de la série 18F, nous pouvons écrire :

```
        LFSR     FSR0,0x000
NEXT    CLRF     POSTINC0
        BTFSS    FSR0H,0
        GOTO     NEXT
```

À l'examen, on voit que la ligne 1 commence par la nouvelle instruction LFSR. Grâce aux instructions standard MOVLW et MOVWF, vous pouvez assigner séparément des valeurs à FSR0L et FSR0H. Toutefois, vous pouvez également initialiser ces deux registres en une fois à l'aide de l'instruction LFSR suivie de FSR0 (ou 1 ou 2), puis d'une virgule suivie d'un nombre hexadécimal de trois chiffres. Le premier chiffre sera placé dans les quatre bits de poids faible de FSR0H et les deux derniers dans FSR0L. Ainsi notre première ligne aura pour effet de faire pointer FSR vers le premier registre de notre espace RAM.
La ligne suivante est une étiquette, NEXT, suivie du familier CLRF. L'opérande POSTINC0 provoquera l'effacement du registre à l'adresse 000 et l'incrémentation du registre FSR0.
Après l'exécution de l'instruction

```
GOTO          NEXT
```

la valeur contenue en 0x001 sera effacée et cette séquence sera réexécutée jusqu'à ce que l'instruction

```
BTFSS FSR0H,0
```

provoque la sortie de la boucle lorsque la valeur de FSR0 atteindra 0x100. La dernière adresse mémoire à être ainsi effacée sera 0x0FF.
Avec la série PIC18F, l'ensemble des 256 octets du banc 0 a été effacé avec seulement 4 lignes de code. Pour effacer la même quantité de mémoire sur la série PIC16, nous aurions du répéter les six lignes concernées deux fois de plus pour les bancs 1 et 2, ce qui aurait fait un total de dix-huit lignes.

Le mois prochain ...

Le deuxième article de cette série s'intéressera aux améliorations en matière d'adressage de la ROM et à l'implémentation des grandes tables de recherche.

(040036-1)

Pointeur vers la Toile

Fiches de caractéristiques techniques PIC18F242/252/442/452 à l'adresse :
http://ww1.microchip.com/downloads/en/DeviceDoc/39564b.pdf