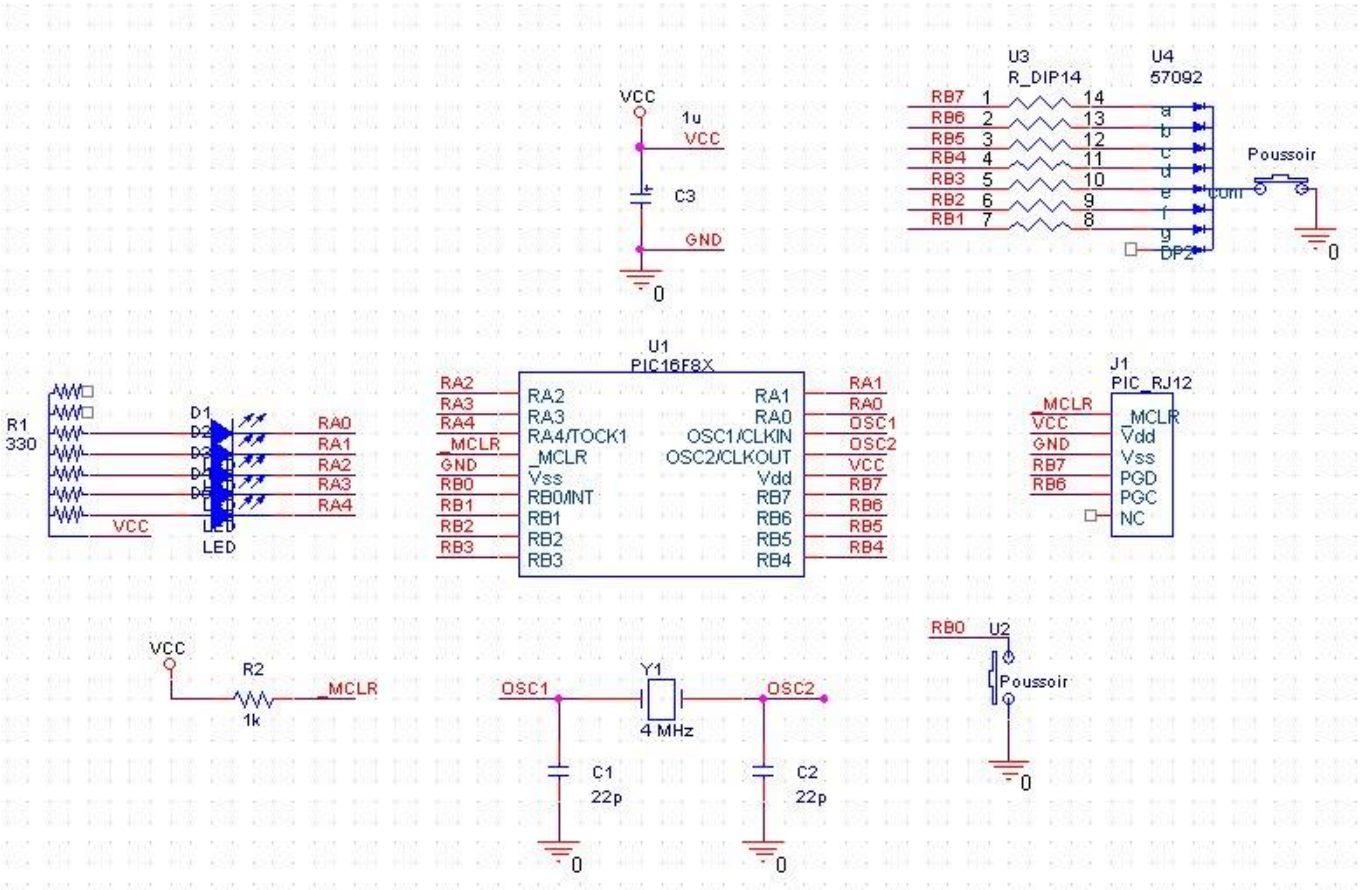
ANNEXE 1 : schéma de la carte



ANNEXE 2 : Présentation des PICs

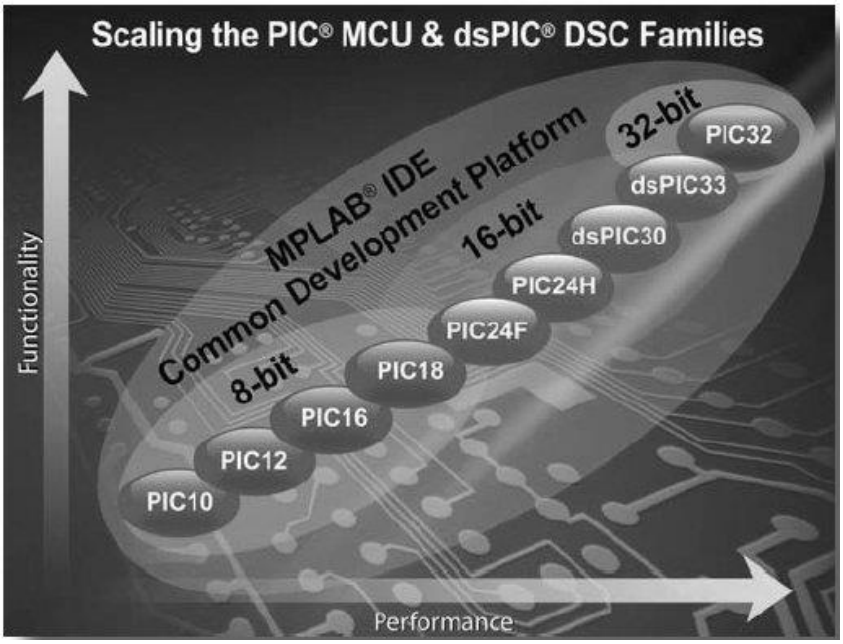
I) DESCRIPTION

II) PROGRAMMER AVEC LES PICS

III) ETUDE DU PIC 16F84

MICROCHIP propose une gamme importante de microcontrôleurs sous l'appellation "PIC", depuis le début des années 90. Il a bâti son succès grâce à son PIC 8 bits, disponible en de multiples versions, permettant ainsi à l'utilisateur de disposer d'un petit microcontrôleur RISC avec des ressources internes répondant au mieux à son application, pour un prix taillé au plus juste (à partir de 0,3 E !). On trouve ainsi des PICs incorporant des ressources élémentaires telles que EEPROM, *timers*, contrôleurs de bus série, etc …, mais aussi des modules plus évolués permettant le contrôle d'écran tactile, la transmission de données par liaison RF…
Les PICs peuvent aussi intégrer des fonctions analogiques telles que CAN, ampli OP, référence de tension, PLL, oscillateurs. Il n'est donc pas étonnant de retrouver ces produits dans bon nombre de systèmes embarqués grand public.

MICROCHIP a progressivement étoffé sa gamme de produits avec des microcontrôleurs 16 et 32 bits ainsi que des DSP et microprocesseurs.



Il fabrique également un grand nombre de circuits intégrés (capteurs, fonctions analogiques variées…) destinés à être interfacés avec ses microcontrôleurs… (voir illustration du constructeur en dernière page). En 2016 Microchip a racheté son principal concurrent, ATMEL avec ses microcontrôleurs AVR, étendant encore un peu plus sa gamme de références. Le site du constructeur www.microchip.com illustre bien la diversité de ses produits et la volonté de proposer des solutions "clés en main" dans un domaine d'application donné.

Dans cette présentation, nous nous limiterons à l'étude des PICs 8 bits et plus précisément la famille PIC16 utilisée dans le cadre de l'activité pratique du module EN111 : 16F84a pour la partie initiation, et 16F877a en ce qui concerne le projet.

I) DESCRIPTION

Il y a plusieurs centaines de références de PICs 8 bits, disponibles en boitiers de 6 à 100 broches. Cette catégorie, aujourd'hui encore la plus répandue, couvre en effet un grand domaine d'applications par la variété des ressources internes disponibles, limitées tout de même en performance par la puissance de calcul restreinte du processeur 8 bits.

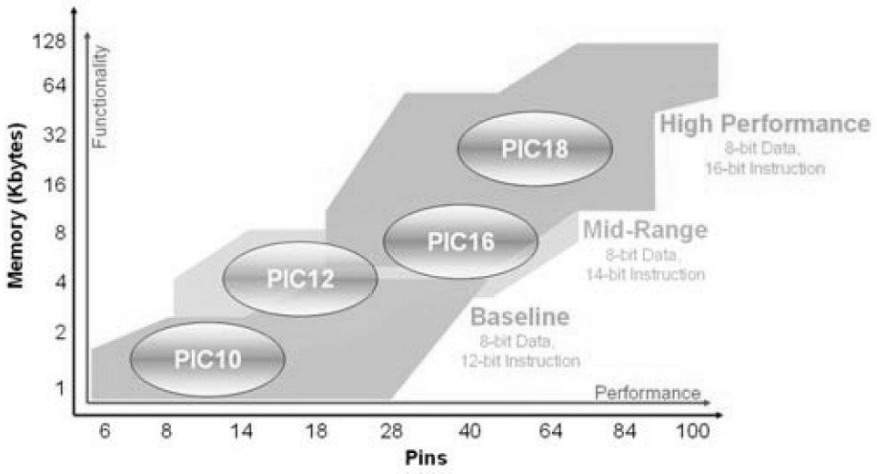
1) Architecture

Les PICs dits "8 bits" sont bâtis autour d'un processeur 8 bits, c'est-à-dire que les registres et le bus de données internes ont un format de 8 bits. Ils ont une architecture Harvard et possèdent donc un bus "programme" distinct du bus de données ce qui leur permet d'accéder et manipuler une donnée et simultanément aller lire l'instruction suivante. Ce parallélisme est possible grâce à un pipeline à 2 niveaux (voir section III-5 "déroulement d'une instruction").

Le processeur est de type RISC (Reduced Instruction Set Computer) et ne possède donc que peu d'instructions (35 pour les 2 PICs que nous utiliserons). Cette caractéristique permet de les coder dans un seul emplacement mémoire pour chaque instruction, ce qui permet un accès et un décodage plus rapide. Une instruction s'exécute ainsi en 1 cycle machine soit 4 périodes d'horloge. Les PICs permettent ainsi une cadence de plusieurs MIPS (Million Instructions Per Second), jusqu'à 5 MIPS pour la série PIC16 que nous utiliserons. La contrepartie de ce gain de rapidité se situe au niveau de la simplicité des instructions qui nécessite d'en associer souvent plusieurs pour réaliser une opération même basique. Un autre inconvénient concerne l'accès aux registres plus complexe puisqu'on ne code qu'une partie de leur adresse dans le champ de l'instruction afin de limiter la taille des emplacements de la mémoire programme, l'autre partie définissant une page mémoire ("banque") sélectionnée au moyen d'un registre spécifique.

2) Différentes familles

Les PICs 8 bits sont découpés en 4 séries (PIC10, PIC12, PIC16 et PIC18) classées par "puissance" en termes de ressources internes, d'entrées/sorties et, ce qui est étroitement lié, en capacité de mémoire programme d'autant plus étendue qu'il y aura de ressources à contrôler. Les PICs ont une pérennité réduite afin de répondre aux besoins du marché, certaines gammes disparaissant (PIC14, PIC16C5…) remplacées par de nouvelles généralement compatibles et plus performantes. Il est bien clair que les PICs sont avant tout destinés aux produits grands publics à faible durée de vie plutôt qu'aux applications militaires…

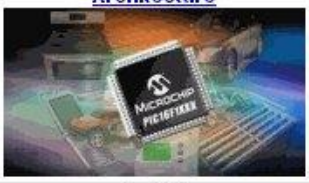


MICROCHIP a défini 3 classes d'architecture pour ses PICs 8 bits, qui se distinguent par la taille des instructions :

Baseline : instructions codées sur 12 bits
Mid-Range et *Enhanced Midrange* : instructions codées sur 14 bits
*High performance (*ou *high end)* : instructions codées sur 16 bits

Une taille plus élevée permet un plus grand nombre d'instructions mais surtout d'augmenter le champ alloué à l'adresse du registre à manipuler et donc de disposer de plus de registres. Ceux-ci sont effet d'autant plus nombreux qu'il y a de ressources (entrées/sorties, *timer*…) à contrôler.

8-bit PIC® Architectures

	Baseline Architecture	Midrange Architecture	Enhanced Midrange Architecture	PIC18 Architecture
				
Pin count	6 – 40	8 – 64	8 – 64	18 – 100
Interrupts	No	Single Interrupt Capability	Single Interrupt Capability with Hardware Context Save	Multiple Interrupt Capability w Hardware Context Save
Operating Performance	5 MIPS	5 MIPS	8 MIPS	10 – 16 MIPS
Instructions	33, 12-bit instructions	35, 14-bit instructions	49, 14-bit instructions	75 - 83, 16-bit instructions
Program Memory	Up to 3 KB	Up to 14 KB	Up to 56 KB	Up to 128 KB
Data Memory	Up to 138 Bytes	Up to 368 Bytes	Up to 4 KB	Up to 4 KB
Features	• Smallest form factor • Lowest cost • Ideal for battery operated or space constrained applications • Easy to learn & use	• Optimal cost-to-performance ratio • Integrated peripherals including SPI, I²C™, UART, LCD, ADC	• C-code Optimized • Enhanced 16 Level Hardware Stack • Enhanced Indirect Addressing • Reduced Interrupt Latency • Simplified Memory Map	• 32 level deep stack, 8x8 hardware multiplier • C-code optimized • Advanced peripherals including CAN, USB, Ethernet touch sensing, and LCD drive
Families	Includes PIC10, PIC12 and PIC16	Includes PIC12 and PIC16	Includes PIC12F1xxx & PIC16F1xxx	PIC18 J-series for cost-sensitive applications wi high levels of integration

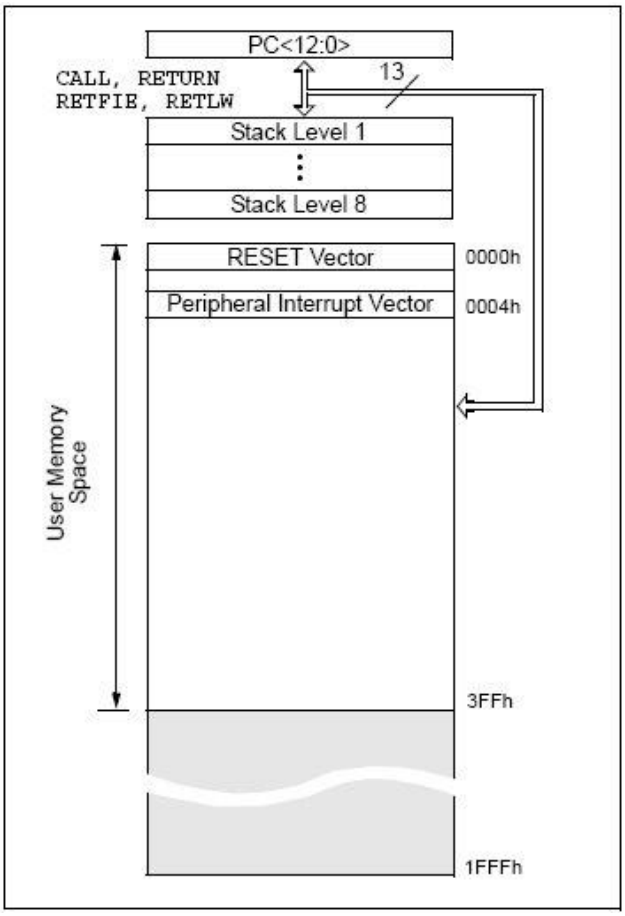
Les 2 PICs étudiés dans le module EN111 font partie de la famille intermédiaire *mid-range* qui s'adresse essentiellement aux produits grand public. Nous nous limiterons à l'exemple du PIC16F84 pour définir l'espace mémoire de programme et de données, les autres références étant calquées sur la même organisation, avec plus ou moins d'emplacements.

3) Espace mémoire

a) Mémoire code programme

Il s'agit d'une mémoire de type FLASH (= EEPROM à accès rapide) sur les PICs de type F tel que le 16F84. On trouve aussi des versions industrielles OTPROM (One Time Programming ROM), et ROM (type CR) programmé en usine par MICROCHIP.

Mémoire programme PIC 16F84



Le PIC 16F84 possède une mémoire FLASH de 1 Kmots (= 1024 mots) de 14 bits et l'espace maximum adressable pour la famille *midrange* est de 8 Kmots (cas du 16F877). Le compteur programme PC (Program Counter) est donc un registre de 13 bits dont seuls les 10 bits de poids faibles sont utiles sur le 16F84. Le registre PC contient à tout instant l'adresse de l'instruction qui sera exécutée au prochain cycle d'horloge.

La pile ("Stack") est une zone mémoire qui sert à stocker des adresses utiles au déroulement du programme. Par exemple lorsqu'on effectue un appel de sous-programme (ou appel de fonction en langage C) l'adresse de l'instruction courante est stockée automatiquement dans la pile permettant ainsi de reprendre le déroulement du programme après cette instruction une fois le sous-programme exécuté. Le processeur effectue pour cela des copies PC→pile (appel) ou pile→PC (retour). La pile est une mémoire à accès séquentiel de type LIFO (Last In First Out) dont le nom fait référence à une pile d'assiette : la dernière posée sera la première à être reprise. Cette zone mémoire n'est pas accessible par l'utilisateur sur les PICs, et ne contient que 8 emplacements ce qui est très peu. Toute récursivité est pour cela interdite par les compilateurs C. La pile étant ici circulaire, la 9^{e} écriture écrase la 1ère définitivement perdue.

2 adresses sont particulières pour les PICs :

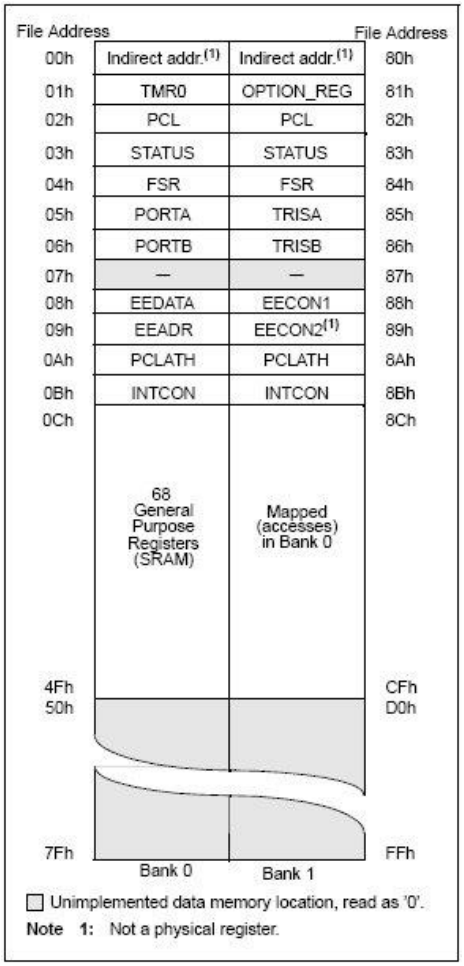
0000h : adresse de l'instruction qui sera exécutée à la mise sous tension ou après un RESET
0004h : adresse de l'instruction qui sera exécutée si une interruption survient (voir partie …)

Remarque : Microchip utilise le terme "vecteur" pour désigner ces 2 emplacements ce qui est un abus de langage car un vecteur est en principe un pointeur en microinformatique, c'est-à-dire qu'il contient l'adresse de la case mémoire qu'il pointe. Les PICs accèdent à ces 2 adresses de la même manière que pour tout autre emplacement en mémoire programme, c'est-à-dire qu'il décode la valeur lue pour obtenir une instruction. En pratique, on place à ces 2 adresses une instruction *GOTO add* où add désigne l'adresse des routines RESET et ISR (Interrupt Service Routine).

Certains PICs plus haut de gamme (à partir de la série PIC24 – MCU de 16 bits) possèdent une table de vecteurs dédiées aux interruptions, permettant ainsi l'exécution d'autant de fonctions d'interruptions différentes liées à des sources de déclenchement spécifiques, chaque vecteur pointant sur l'adresse du point d'entrée de la fonction correspondante.

b) Mémoire données (8 bits)

① RAM **Mémoire de données du PIC 16F84**



Comme on peut le voir avec l'exemple du PIC 16F84 la RAM est divisée en pages mémoires de 128 octets (soit 7 bits d'adresse) que MICROCHIP désigne par le terme "banques" (2 banques BANK0 et BANK1 sur le PIC16F84). C'est une des grandes particularités des PICs et une conséquence directe du type RISC de leur processeur intégré dans une architecture à faible coût. En effet, les instructions sont peu nombreuses dans le but de chacune pouvoir les coder sur un seul emplacement de la mémoire programme, et la taille commune de toutes instructions a donc été volontairement restreinte (14 bits sur le PIC16F84) dans un soucis d'économie et tout en respectant les exigences de l'ensemble des instructions. Cela induit par contre une limitation sur l'espace (champ de 7 bits sur le PIC16F84) alloué dans le code de l'instruction pour spécifier l'adresse d'un registre utilisé comme argument.

Les registres sont donc divisés en 2 banques (voire 4 sur certains PICs) sélectionnables par des bits supplémentaires qui correspondraient aux bits de poids fort de l'adresse complète : RP0 (+RP1) du registre **STATUS** ou directement par un registre dédié (BANK REGISTER sur les PICs de la famille *High performance).*

La présélection de la bonne banque est donc une contrainte à ne pas oublier avant l'accès à un registre en RAM sous peine de ne pas manipuler le bon registre !

La RAM contient les registres à fonction dédiée (SFR) qui permettent le contrôle du processeur ou des périphériques (les SFRs les plus utilisés sont situés en BANK0 qui est la banque de base "par défaut"). A la suite de ces registres se trouve de la RAM utilisable pour stocker des variables "GPR" (General Purpose Registers).

On peut remarquer que certains registres SFR sont accessibles dans toutes les banques (le *STATUS register* puisqu'il permet de changer de banque + quelques autres), c'est aussi le cas pour les 68 GPRs du PIC16F84 (attention, ce n'est pas vrai avec tous les PICs).

② EEPROM (selon circuit)

Certains PICs disposent d'une mémoire EEPROM pour stocker des constantes accessibles au moment de la programmation mais aussi par le programme. Attention, cette dernière procédure est assez lourde (surtout en écriture) et le temps d'accès en écriture est relativement long (≈ 10 ms). Ces octets ne sont donc pas à utiliser pour stocker des variables temporaires évoluant souvent dans le programme.

Le PIC 16F84 dispose de 64 octets situés à partir de l'adresse 0x2100

II) PROGRAMMER AVEC LES PICS

1) Principaux registres

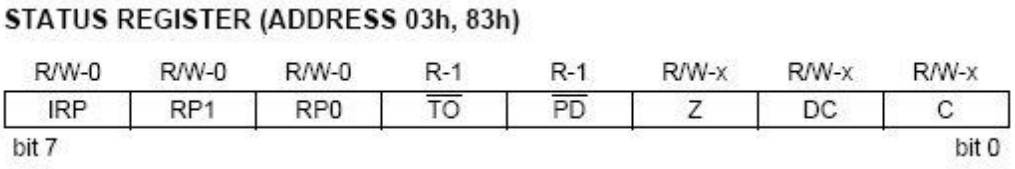
Certains registres sont indispensables au déroulement du programme.

a) Le registre de travail **W** (Working register)

Le processeur ne peut déplacer une donnée ou effectuer une opération arithmétique ou logique sur un registre (SFR ou GPR) qu'en utilisant le registre de travail W. C'est un registre 8 bits indépendant des registres SFR et qui est directement associé à l'ALU (Arithmetic Logic Unit). On le qualifie aussi d'*accumulateur* car, comme on peut le remarquer sur le schéma de l'architecture interne (cf partie III-3 pour le PIC16F84a), il permet de réinjecter directement dans l'ALU le résultat de l'opération précédente ce qui réduit le temps de calcul des opérations portant sur plus de 2 opérandes (addition de 3 nombres par exemple).

b) Le registre d'état **STATUS** (0x03/0x83)

C'est un registre qui fournit de manière générale des indications liées à l'exécution de la dernière instruction par le processeur. On y trouve en particulier les bits d'état qui renseignent sur la valeur de la dernière opérande (=donnée manipulée par une instruction). Comme déjà évoqué, ce registre contient également les bits qui permettent de sélectionner les différentes pages mémoires en RAM ("banques").



C (b0) : Carry (report). Bit de retenue, correspond au 9ème bit de l'opération.

DC (b1) : Digit Carry. Utilisé lorsque l'on travaille avec des nombres BCD : il indique un report du bit 3 vers le bit 4 dans une opération.

Z (b2) : Zero. Vaut 1 si le résultat de la dernière opération vaut 0.

/PD (b3) : Power down Indique quel événement a entraîné le dernier arrêt du PIC (instruction sleep ou dépassement du temps du watchdog).

/TO (b4) : Time-Out bit Indique (si 0), que la mise en service suit un arrêt provoqué par un dépassement de temps ou une mise en sommeil. Dans ce cas, /PD effectue la distinction.

RP0 (b5) : Register Bank Select 0 Sélectionne la banque de RAM qui sera utilisé au prochain accès (0 = banque 0).

RP1 (b6) : Register Bank Select 1 Permet la sélection des banques 2 et 3. Inutilisé pour le 16F84, il doit être laissé à 0 pour garantir la compatibilité ascendante (portabilité du programme).

IRP (b7) : Indirect RP Permet de décider quelle banque on adresse dans le cas de l'adressage indirect pour les PICs disposant de plus de 2 banques (inutile pour le 16F84).

c) Le compteur programme **PC**

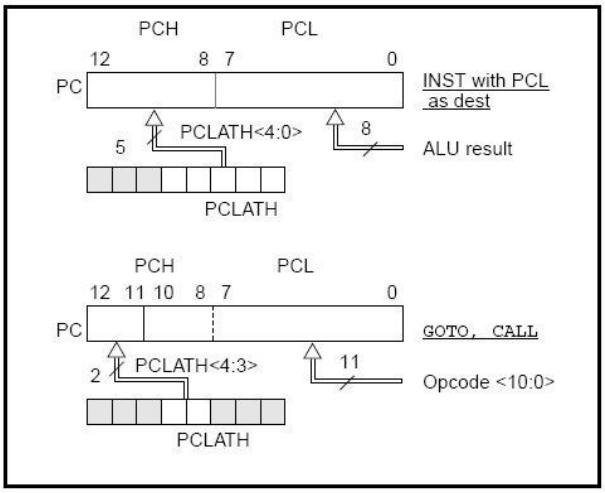
Il pointe sur la prochaine instruction à exécuter. Comme il est codé sur 13 bits (accès à toute la mémoire de codes programmes) il nécessite d'être représenté par 2 registres :

PCL	(PC Low)	(0x02/0x82) :	8 bits de poids faible
PCLATH	(PC LATch counter High)	(0x0A/0x8A) :	5 bits de poids fort b4 à b0

NB : pour le 16F84, 1Kmots de mémoire programme ⇒ b4, b3, b2 de PCLATH non utilisés.

Le compteur programme est normalement incrémenté d'une unité après l'exécution de chaque instruction du programme, sauf si celle-ci agit directement sur lui-même. C'est le cas avec les instructions de test (modification automatique du PC selon résultat du test) et les instructions de branchement qui spécifient dans leur argument l'adresse de la prochaine instruction qui devra être exécutée :

Chargement du PC dans différentes situations



Comme le montre cette figure, le compteur programme peut en effet être modifié directement par des instructions spécifiques telles que les déroutements (GOTO) ou les appels de sous-programme (CALL). Mais attention, la totalité des bits définissant la nouvelle adresse du PC ne pouvant être codée en une seule fois dans les 14 bits d'une instruction, il faudra peut-être définir les bits de poids forts par un accès spécifique à PCLATH.

2) Modes d'adressage

Le mode d'adressage désigne la manière de désigner la donnée à manipuler (opérande). Il n'existe que 3 modes d'adressage sur les PICs, qui sont les plus usuels sur les processeurs :

➢ literal (= immédiat) : la valeur de l'opérande est directement donnée dans l'instruction

ex : MOVLW k *charge la valeur k dans W*

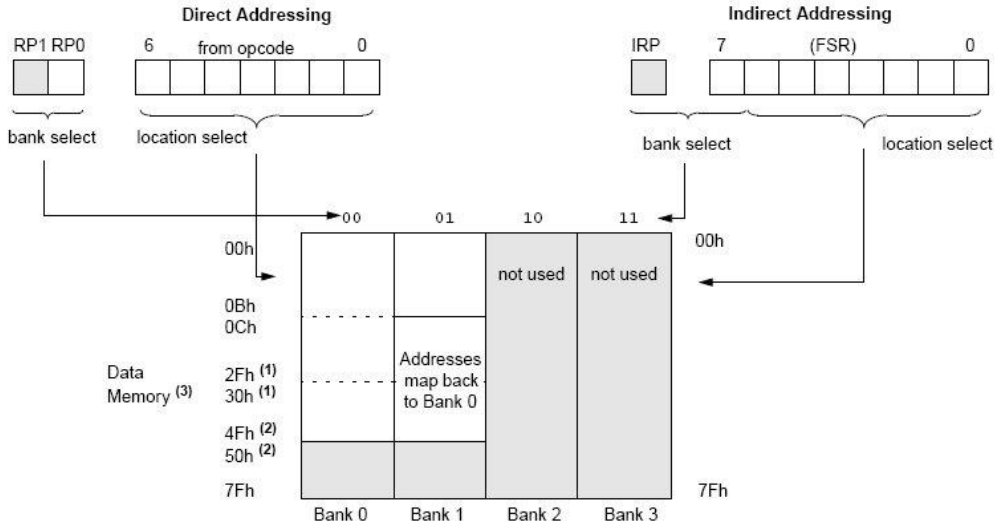
A noter que ce mode d'adressage n'est utilisable qu'avec W car la taille réduite des instructions ne permet pas de coder sur ses 14 bits l'adresse d'un registre en RAM en plus des 8 bits de la donnée k (cf partie suivante)

➢ direct : l'opérande est le contenu de l'adresse (registre) spécifié

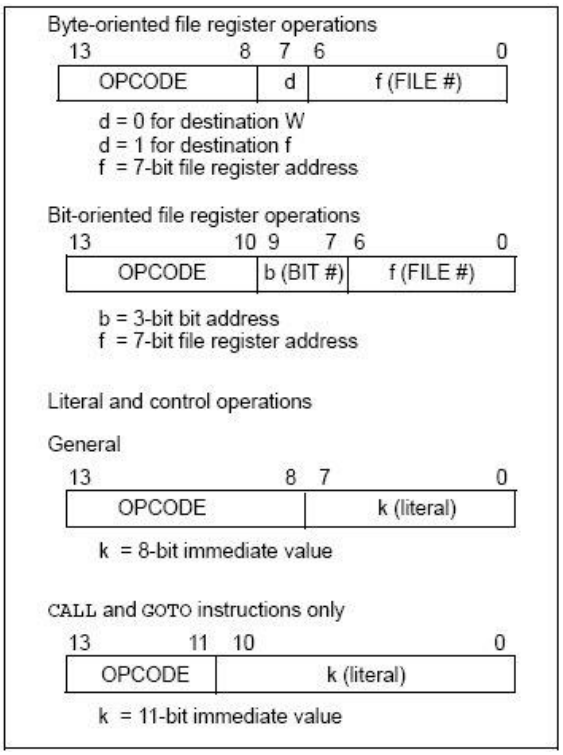
ex : MOVWF 0x05 *charge le contenu de W dans l'adresse 05h*
(= registre PORTA si banque 0)

➢ indirect : l'opérande est le contenu de la case mémoire dont l'adresse est elle-même contenue dans le registre spécifié (= pointeur en langage C). Sur les PICs l'utilisation de ce mode d'adressage est particulier : le pointeur est unique, il s'agit du registre FSR (0x04/0x84). Pour utiliser ce mode d'adressage on fait appel dans l'instruction à un registre spécial, le registre INDF (0x00/0x80) qui n'existe pas vraiment.

ex : MOVLW 0x05 *charge la valeur 05h (adresse de PORTA) dans W*
MOVWF 0x04 *transfert la valeur 05h dans le registre FSR*
(on a ainsi initialisé le pointeur FSR sur PORTA)
MOVF INDF,W effectue l'accès indirect via le pointeur FSR
(*le contenu de PORTA est copié dans W)*



3) Types d'instruction



a) Instruction qui porte sur 1 octet : manipule l'octet contenu dans le registre *f* dont l'adresse est spécifiée sur 7 bits (= taille du bus d'adresse de la RAM) en accord avec la banque sélectionnée.

Remarque : le bit *d* (destination) désigne le registre qui sera utilisé pour stocker le résultat de l'opération avec seulement 2 possibilités : soit W (si d=0), soit le registre source *f* (si d=1) puisque la taille réduite de l'instruction ne permet pas de définir l'adresse d'un registre supplémentaire.

b) Instruction qui porte sur un bit : manipule directement le bit numéro *b* parmi les 8 bits d'un registre *f* spécifié.

c) Instruction literal : manipule des données codées dans l'instruction même (adressage immédiat).

d) Sauts et appel de sous-programme : 3 bits pour coder l'instruction, les 11 bits restant pour coder l'adresse de destination. Si la mémoire programme est supérieure à 2 Kmots (16F877…) on positionne les bits 3, 4 de **PCLATH**.

4) Jeu d'instructions (exemple de la famille *mid-range* avec 35 instructions)

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

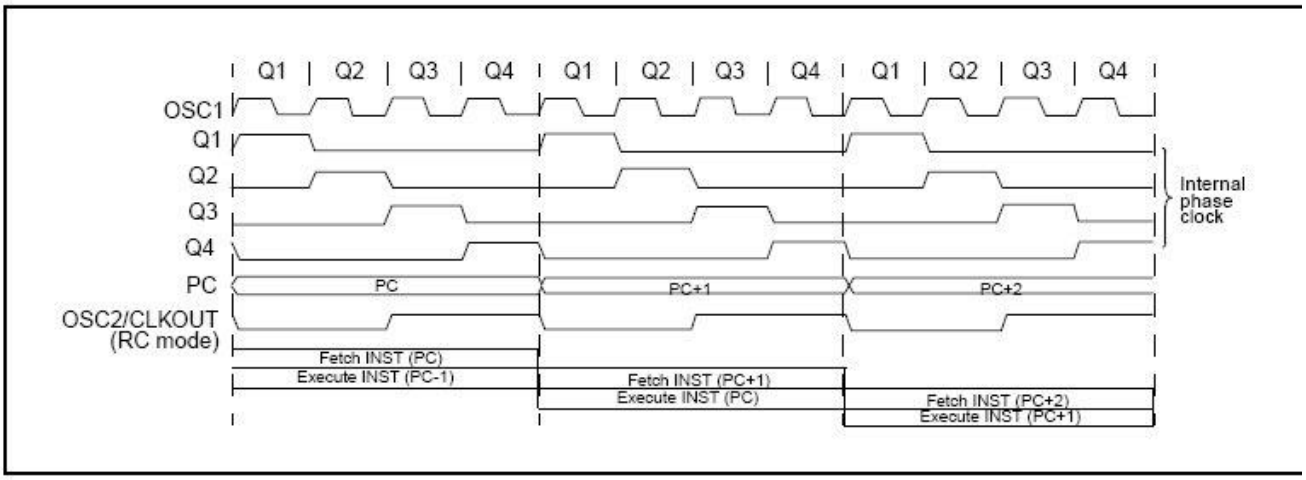
3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compat-ibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

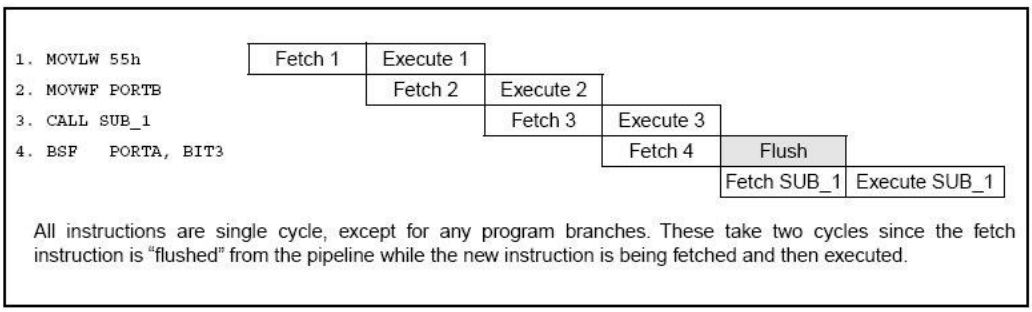
5) Déroulement d'une instruction

La fréquence du signal d'horloge externe est divisée par 4 à l'intérieur du PIC pour obtenir 4 horloges internes en quadrature nommées Q1, Q2, Q3 et Q4 qui délimitent en fait les 4 phases de l'exécution d'une instruction. Pour commencer le PC est incrémenté par Q1 et l'instruction est ensuite chargée dans le registre instruction par Q4 (délai nécessaire à cause du temps d'accès de la mémoire programme). Le cycle suivant effectue l'exécution de l'instruction avec, dans le cas le plus fréquent :

Q1 : décodage de l'instruction
Q2 : lire l'opérande (W, registre en RAM, donnée immédiate…)
Q3 : effectuer un éventuel traitement (addition, …)
Q4 : stocker le résultat (dans l'opérande source, W…)



Une instruction nécessite donc 1 cycle pour que le PIC aille la chercher en mémoire programme et un autre pour l'exécuter. Un pipeline à 2 niveaux permet de mener ces 2 types d'opération en parallèle : dans un même cycle de 4 périodes d'horloge, le PIC va chercher l'instruction *n* tout en exécutant l'instruction *n-1* pré-chargée durant le cycle précédent. Une instruction s'exécute donc en un seul cycle, sauf lorsqu'il y a déroutement (changement du PC par une instruction, cf exemple ci-dessous) où il faut rajouter un cycle "mort" pour charger la nouvelle instruction avant son exécution.



III) ETUDE DU PIC 16F84

MICROCHIP

PIC16F84A

18-pin *Enhanced* FLASH/EEPROM 8-Bit Microcontroller

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

Peripheral Features:

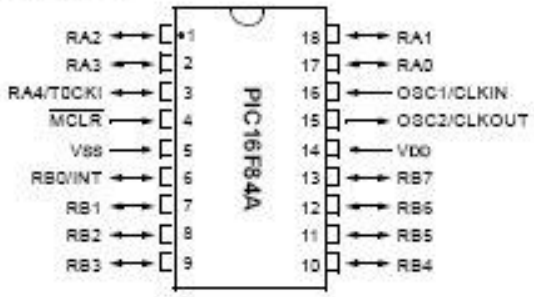
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

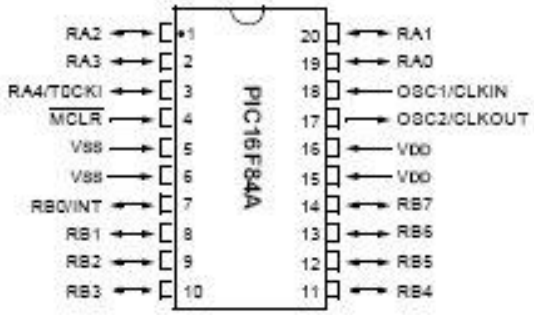
- 10,000 erase/write cycles *Enhanced* FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Code protection
- Power saving SLEEP mode
- Selectable oscillator options

Pin Diagrams

PDIP, SOIC



SSOP



CMOS Enhanced FLASH/EEPROM Technology:

- Low power, high speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

1) Description rapide

On s'intéresse ici au PIC 16F84A qui remplace le 16F84 en tout point identique mais avec une mémoire FLASH améliorée (10000 écritures possibles au lieu de 1000).

Résumé des caractéristiques du PIC 16F84 (A)

- Boitier 18 broches (DIP ou SOIC)
- Alimentation : (2 V < < 5,5 V)
- Fréquence d'horloge ≤ 20 MHz (quartz ou circuit RC selon programmation)
  (1 cycle d'horloge = 4 périodes d'horloge)

- Mémoire programme : 1 Kmots de 14 bits (bit 3 et 4 de PCLATH inutilisés)
  de type FLASH (16F84) ou EEPROM (16C84)

- Mémoire de données :
  - RAM : 2 banques de 128 octets dont certains sont identiques (!) et d'autres inutilisables.
    → 15 registres SFR
    → 68 octets libres pour stocker des variables
  - EEPROM : 64 octets

- Périphériques :
  - 2 ports d'entrée/sortie A (5 bits)
    B (8 bits)
  - 1 *timer* de 8 bits (avec prédiviseur 8 bits programmable)
  - 1 chien de garde

- 4 sources d'interruption (1 seul "vecteur")
- pile à 8 niveaux
- mode *standby*

2) Registres SFR

En plus des registres principaux décrits dans la section II-1, on trouve des registres dédiés à la gestion des périphériques internes associés au processeur (cf schéma *architecture interne* section III-3).

Le tableau ci-dessous récapitule tous les registres du PIC16F84 (A), avec la désignation de chacun de leurs 8 bits ainsi que leur état au démarrage et après un reset. Ces différents registres vont être décrits dans les sections suivantes.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

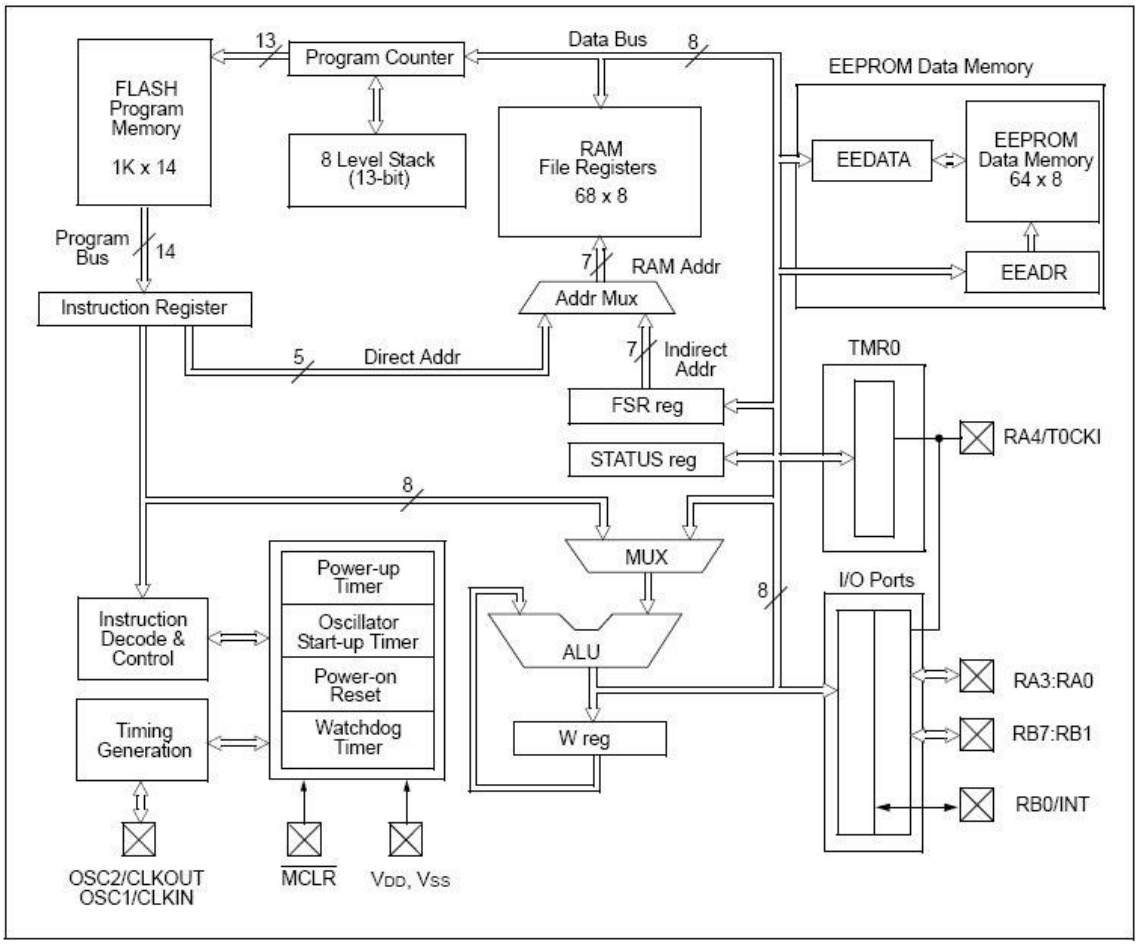
Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

3) Architecture interne



On observe sur ce schéma toutes les mémoires liées au µP (sur la partie supérieure : programme FLASH, pile, RAM de données), les périphériques internes (à droite : EEPROM, TIMER0, PORTs A et B), les ressources de contrôle (sur la gauche : le registre + décodeur d'instruction, le PC), l'unité de traitement (ALU + accumulateur W) et les blocs gérant le système (en bas : générateur d'horloge, circuits de démarrage, chien de garde…)…n'apparaît pas le cœur du processeur avec tous les blocs séquentiels liés aux instructions.

4) Les ports d'entrée/sortie A et B

Un port permet une interface entrée/sortie avec des périphériques externes au PIC. Il s'agit schématiquement d'un registre qui sert de tampon entre le bus de données et les broches du composant. Chaque port est géré par les registres :

- **PORT** (**A**:0x05 , **B**:0x06) qui est le "contenu" du port correspondant (= image de l'état des bits du port).

- **TRIS** (**A**:0x85 , **B**:0x86) qui configure indépendamment les bits du port en entrée (1) ou en sortie (0).

ex : TRISB = 0x0F *PB7, PB6, PB5, PB4 en sortie*
PB3, PB2, PB1, PB0 en entrée

➢ le registre commun **OPTION** (0x81) qui permet grâce au bit RBPU (b7) de mettre en service (si RBPU = 0) les résistances de rappel à + 5V (pull up) sur tous les bits du port B.

NB : - Certaines broches de ces ports ont une double fonction. Par exemple RB0/INT peut aussi servir pour provoquer une interruption externe (matérielle).

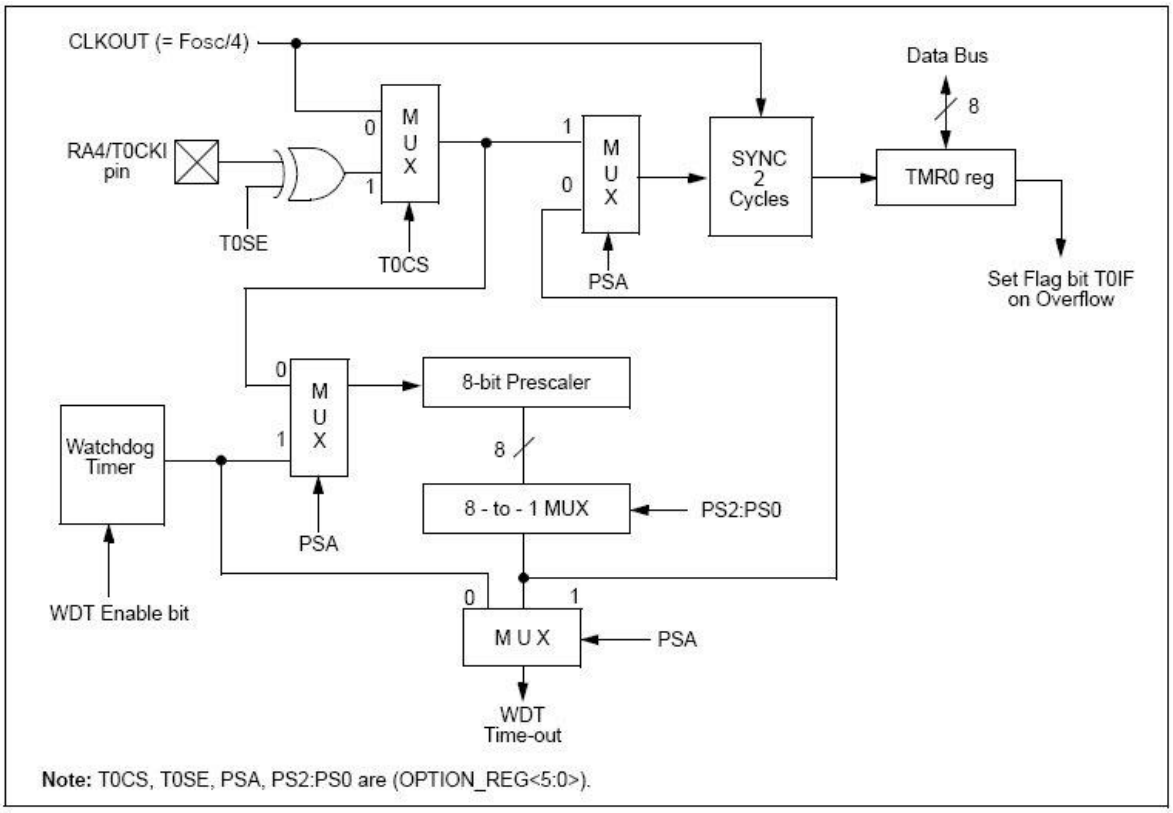
- D'un point de vue électrique, il y a aussi des différences entre les broches et selon les PORTs. Par exemple RA4 a une sortie en drain ouvert (nécessité de placer une résistance externe de pull up), consulter la datasheet - section 4 pour plus d'infos.

- Courant max de sortie : 25 mA / broche et total de 50 mA (port A) , 100 mA (port B)

5) Le *timer* TIMER 0

C'est un compteur 8 bits dont la valeur en sortie est accessible par l'intermédiaire du registre **TMR0** (0x01) en lecture mais aussi en écriture. Il peut être incrémenté de 2 façons :

① par les cycles d'horloge du PIC (attention 1 cycle = 4 périodes de H) : c'est le mode TIMER
② par les impulsions reçues sur la broche RA4/T0CKI : c'est le mode COMPTEUR



Le TIMER 0 est lui aussi géré par le registre **OPTION** (0x81) :

➢ le bit T0CS (b5) permet de choisir le mode :
- T0CS = 0 → mode TIMER
- T0CS = 1 → mode COMPTEUR

➢ le bit T0SE (b4) permet de choisir sue quelle transition de la broche RA4/T0CKI le comptage est effectué en mode COMPTEUR :
- T0SE = 0 : comptage si l'entrée RA4/T0CKI passe de 0 à 1
- T0SE = 1 : comptage si l'entrée RA4/T0CKI passe de 1 à 0

➢ les bits PS2 (b2), PS1 (b1), PS0 (b0) permettent de définir le rapport de division du prédiviseur (**prescaler**)

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

NB : Le *prescaler* peut aussi être utilisé pour le chien de garde (voir plus loin) mais il ne peut plus alors être utilisé par le TIMER 0.

➢ le bit PSA (b3) permet d'affecter le *prescaler* : - au TIMER 0 (PSA = 0)
- **ou** au chien de garde (PSA = 1)

NB : - Pour désactiver le *prescaler* il faut l'affecter au chien de garde avec un rapport 1 : 1 (PS2=PS1=PS0=0)
- Les 8 bits du *prescaler* ne sont pas accessibles en lecture ni en écriture

ex : $f_{horloge}$ = 4 MHz *cycle d'horloge de durée = 1 µs*
T0SC = 0 *mode TIMER*
PSA = 0 *prescaler affecté au TIMER 0*
(PS2, PS1, PS0) = 010 *: prescaler = 8*

⇒ TIMER 0 incrémenté toutes les 8 µs

Utilisation du TIMER 0

Le TIMER 0 peut être utilisé de 2 façons :

① Par scrutation du flag T0IF

Lorsque le TIMER 0 déborde (passage de la valeur 255 à 0) le bit T0IF (b2) du registre **INTCON** (0x0B, 0x8B) passe à 1 (et reste à 1 tant qu'il n'a pas été remis à 0 par le programme !). Il suffit donc de scruter le bit T0IF pour savoir si 256 incrémentations ont eu lieu.

Inconvénient de cette méthode : on perd du temps à attendre et tester le TIMER 0. De plus, le programme risque de ne pas réagir assez rapidement (si la fréquence des tests est inférieure à celle de l'incrémentation de TMR0)

Remarque : Si l'on souhaite mesurer une durée correspondant à N incrémentations du *timer* :

N < 256 : on préférera initialiser le registre **TMR0** à la valeur de N codée en complément à 2 plutôt que de scruter la valeur de **TMR0** et de la comparer à N (test de T0IF plus rapide)

N ≥ 256 : on utilise le nombre de boucles nécessaires pour comptabiliser les débordements du *timer*, le reliquat étant obtenu en initialisant de la même façon le registre TMR0 avec le complément à 2 de (N modulo 256)

② Par interruption (voir partie suivante)

Cette méthode résout le problème posé par la précédente car le programme principal n'a pas à tester le TIMER 0 : le débordement du TIMER 0 génère une interruption (si le registre **INTCON** est bien configuré…) qui déroute "aussitôt" le PIC du programme principal pour traiter un sous-programme prévu. C'est donc cette méthode qui est de loin la plus utilisée.

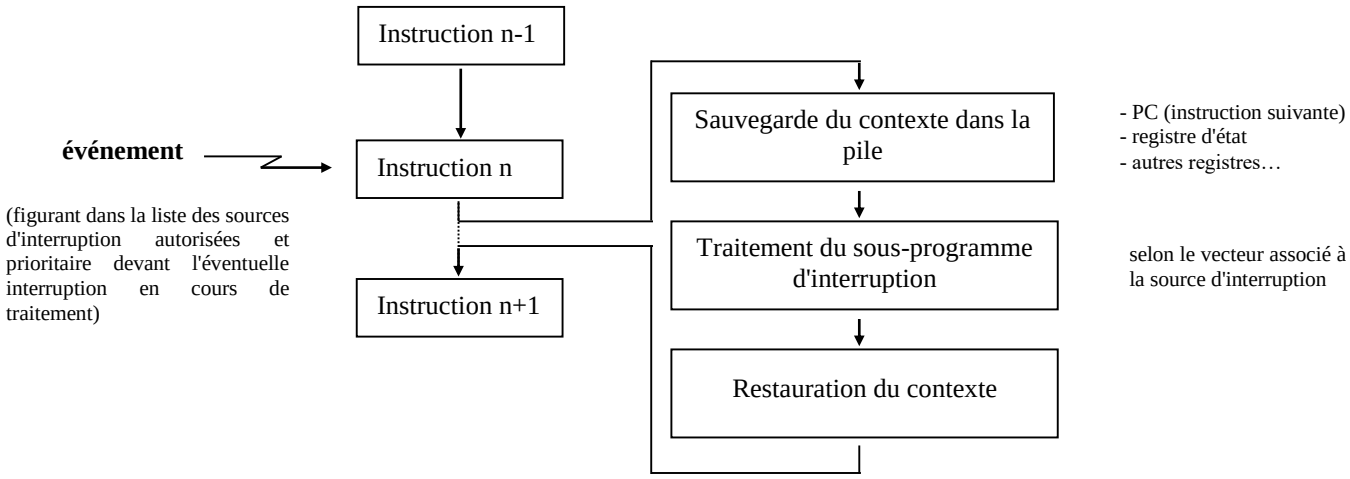
La remarque précédente reste valable.

6) Les interruptions

a) Principe

On parle d'interruption lorsqu'un événement survient et interrompt le déroulement normal du programme afin de pouvoir traiter immédiatement l'événement. Une fois l'événement traité (routine d'interruption terminée) le programme reprend à l'endroit où il avait été interrompu, comme si l'interruption n'avait pas eu lieu.

L'intérêt de ce mécanisme est d'abord de minimiser le temps de réaction du processeur vis-à-vis de l'action à mener en lien avec l'événement, comparativement à une solution classique de "polling" (boucle de test sur l'événement). Le 2^{e} avantage découle du 1er, on libère le processeur de cette phase de scrutation (gain de temps CPU). Le déroulement classique d'une procédure d'interruption a lieu selon le schéma :



b) Mécanisme d'interruption sur les PICs

- Tout d'abord l'adresse de début du sous-programme d'interruption est fixe (un seul vecteur), il s'agit toujours de l'adresse 0x04. Toute interruption provoquera donc le saut du programme vers cette adresse.

- Toutes les sources d'interruption déclenchant l'exécution de la même routine à cette adresse, si le programmeur utilise plusieurs sources d'interruptions, il lui faudra déterminer lui-même laquelle il va devoir traiter.

- Les PICs jusqu'à la série PIC16 ne sauvegardent rien automatiquement en cas d'interruption, hormis bien sûr la valeur du PC indispensable pour revenir au déroulement normal du programme. C'est donc à l'utilisateur de se charger de la sauvegarde des registres (opération par contre prise en charge sur les PICs à partir de la série PIC18.

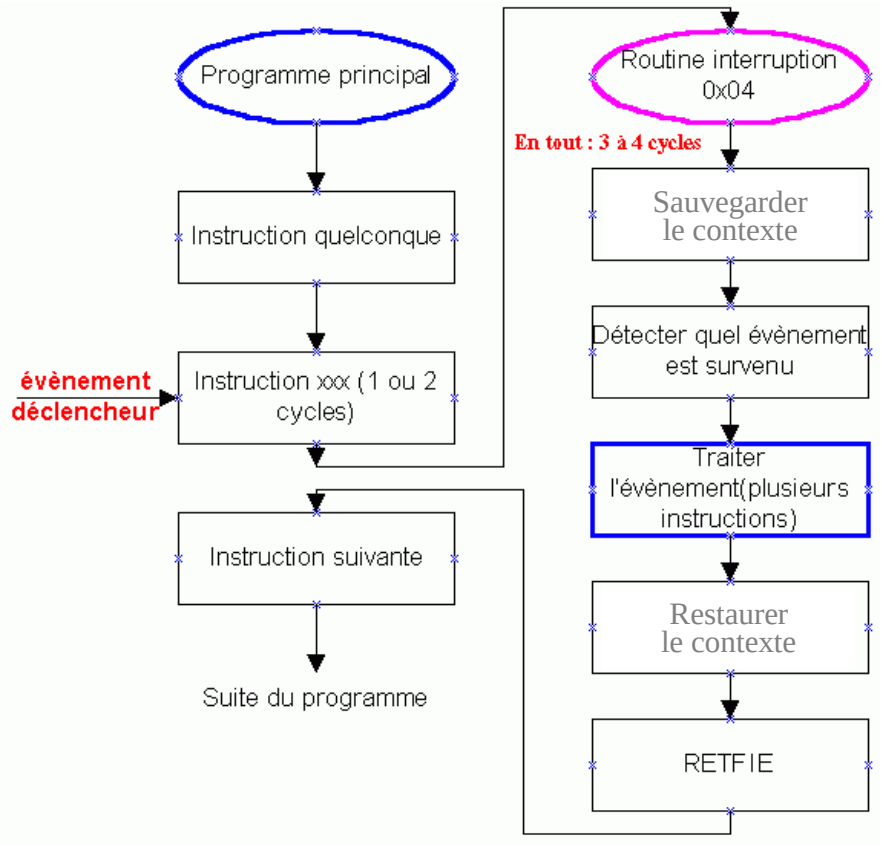
- Le contenu du PC est sauvegardé sur la pile interne (8 niveaux). Donc, si on utilise des interruptions, on ne dispose plus que de 7 niveaux d'imbrication pour les sous-programmes (moins si on utilise des sous-programmes dans les interruptions).

- Le temps de réaction d'une interruption est calculé de la manière suivante : le cycle courant de l'instruction est terminé, le flag d'interruption est lu au début du cycle suivant. Celui-ci est achevé, puis le processeur s'arrête un cycle pour charger l'adresse 0x04 dans PC. Le processeur va à l'adresse 0x04 où il lui faudra un cycle supplémentaire pour charger l'instruction à exécuter. Le temps mort total sera donc compris entre 3 et 4 cycles.

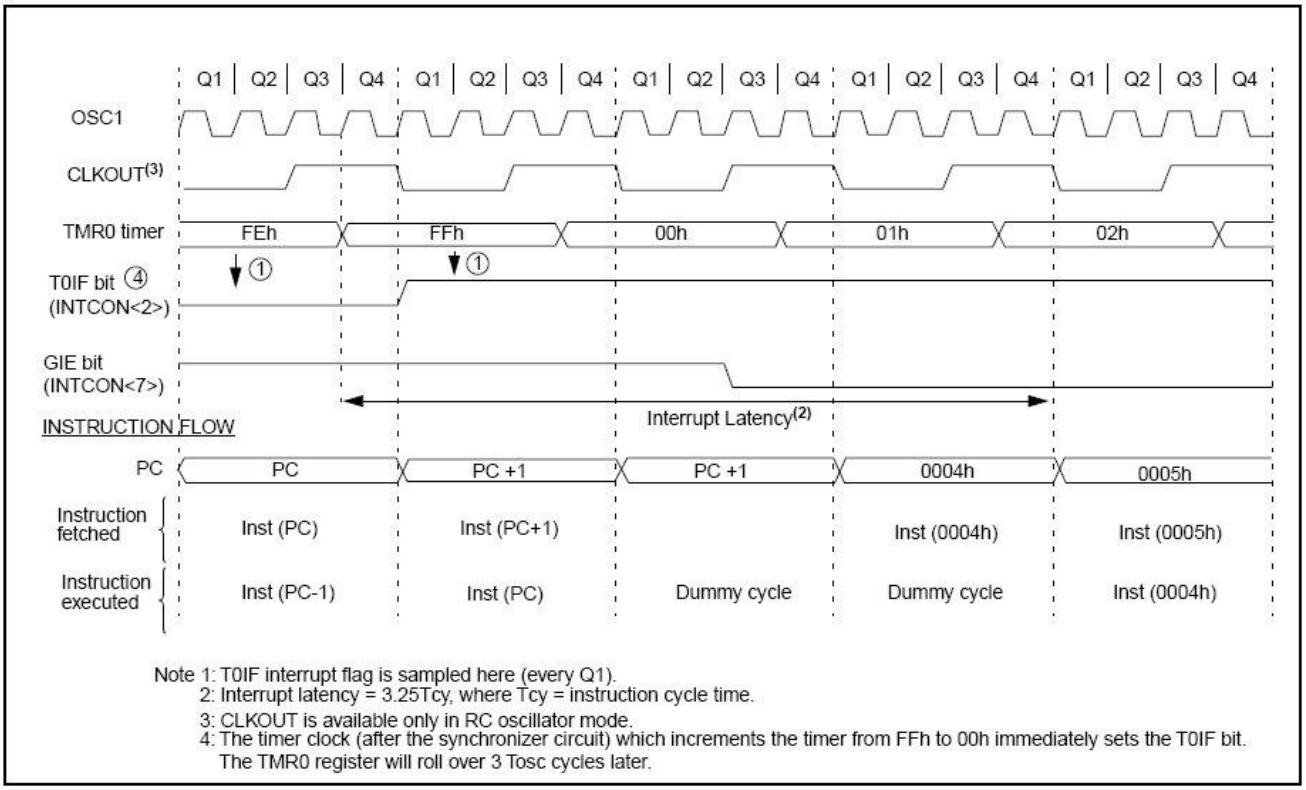
- Une interruption ne peut pas être interrompue par une autre interruption. Les interruptions sont donc invalidées automatiquement lors du saut à l'adresse 0x04 par l'effacement du bit GIE (voir plus loin).

- Les interruptions sont remises en service automatiquement lors du retour de l'interruption. L'instruction RETFIE agit donc exactement comme l'instruction RETURN, mais elle repositionne en même temps le bit GIE.

On obtient ainsi l'organigramme suivant en cas d'interruption sur les PICs :



Exemple du déclenchement d'une interruption par le débordement du TIMER0



c) Les interruptions avec le PIC 16F84

Le 16F84 est très pauvre à ce niveau, puisqu'il ne dispose que de 4 sources d'interruptions possibles (contre 13 pour le 16F877 par exemple). Les événements susceptibles de déclencher une interruption sont les suivants :

- TIMER0 : débordement du TIMER 0. Une fois que le contenu de **TMR0** passe de 0xFF à 0x00, une interruption peut être générée.

- EEPROM : une interruption peut être générée lorsque l'écriture dans une case EEPROM interne est terminée.

- RB0/INT : une interruption peut être générée lorsque la broche RB0, encore appelée INTerrupt pin, étant configurée en entrée, celle-ci reçoit un front. Le front actif est défini par le bit INTEDG (b6) du registre **OPTION** (0x81) :

  INTEDG = 1 → front montant

  INTEDG = 0 → front descendant

- PORTB : de la même manière, une interruption peut être générée si un changement de niveau (front montant ou descendant) se produit sur une des broches RB4 à RB7. Il n'est pas possible de limiter l'interruption à une seule de ces broches. L'interruption sera effective pour les 4 broches ou pour aucune.

Les interruptions sont gérées par le registre **INTCON** (0x0B, 0x8B) :

GIE (b7) : Global Interrupt Enable bit. Il doit être activé pour permettre le déclenchement de toute interruption (il sert surtout à empêcher toute IT lorsqu'il est désactivé).

EEIE (b6) : EEprom write complete Interrupt Enable bit. Ce bit permet d'autoriser une interruption en fin d'écriture en EEPROM (voir paragraphe 7).

T0IE (b5) : Tmr0 Interrupt Enable bit : Autorise une interruption lors du débordement du TIMER 0.

INTE (b4) : INTerrupt pin Enable bit : Autorise une interruption dans le cas où un front actif se présente sur la broche RB0.

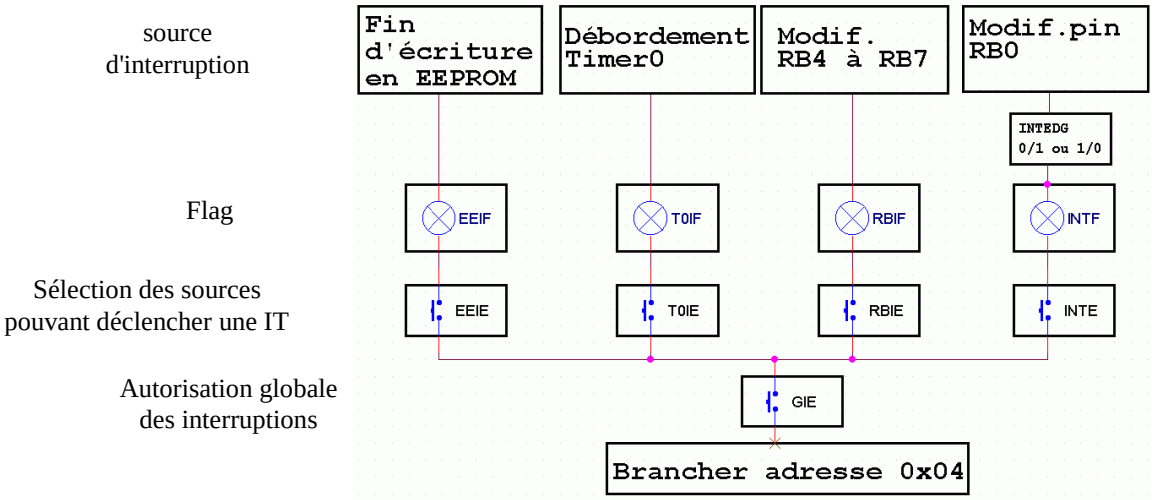
RBIE (b3) : RB port change Interrupt Enable bit : Autorise une interruption s'il y a un changement de niveau sur une des entrées RB4 à RB7.

T0IF (b2) : Tmr0 Interrupt Flag bit : C'est un Flag (indicateur), donc il signale. Ici c'est le débordement du TIMER 0.

INTF (b1) : INTerrupt pin Flag bit : signale une transition sur la pin RB0 dans le sens déterminé par INTEDG (b6) du registre **OPTION**.

RBIF (b0) : RB port Interrupt Flag bit : signale qu'une des entrées RB4 à RB7 a été modifiée.

Le schéma suivant permet de représenter de façon synthétique le rôle des bits du registre **INTCON** :



Remarques :

- Les indicateurs (flags) permettent de connaître la source qui a provoqué l'interruption (attention **ils doivent être remis à 0 par le sous-programme d'IT**).

- Les bits EEIE, T0IE, RBIE, INTE permettent d'autoriser ou non la source correspondante à déclencher une interruption.

- Le bit GIE sert de validation globale et a surtout pour rôle d'interdire tout déclenchement d'IT. Comme les PICs ne peuvent traiter qu'une seule IT à la fois, le bit GIE est automatiquement mis à 0 au début du traitement de l'IT (il est restauré à 1 automatiquement par l'instruction RETFIE)

- Le bit EEIF (indicateur de fin d'écriture en EEPROM) est placé à part, c'est le bit 4 du registre **EECON1** (0x88)

d) Sauvegarde et restauration du contexte

Le PC est sauvegardé dans la pile et restauré automatiquement à la fin de l'IT. Le sous-programme d'IT doit donc gérer la sauvegarde des registres **STATUS** et **W**. On ne peut, comme on le fait en général pour stocker ces registres, utiliser la pile car elle n'est pas accessible. On utilise donc 2 emplacements en RAM.

7) Le chien de garde (Watch Dog)

C'est un mécanisme de protection du programme qui en théorie ne doit pas servir mais qui en pratique peut permettre de sortir d'une boucle d'attente infinie (événement attendu qui ne se produit pas…) ou de vérifier si le programme ne s'est pas égaré. Le Watch-Dog (WD) sert également à réveiller un PIC placé en mode SLEEP (voir paragraphe 6).

a) Description

Il s'agit en fait d'un *timer* (nommé WDT) qui possède sa propre horloge dont la fréquence dépend de la température et de la tension d'alimentation (mais pas de $f_{horloge}$!). Le constructeur donne :

7 ms ≤ périodicité du débordement du *timer* WD = 18 ms typique ≤ 33 ms

Le temps de débordement peut être allongé grâce au *prescaler* (qui fonctionne ici en post-diviseur), si PSA = 1, qui peut multiplier ce temps au maximum × 128 selon le tableau :

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

b) Principe

Chaque fois que l'instruction CLRWDT est exécutée par le PIC, le WDT est remis à 0 ainsi que la valeur contenue dans le prédiviseur. Si, pour une raison anormale, cette instruction n'est pas exécutée avant le débordement du WDT, le PIC est redémarré à l'adresse 0x00 et le bit /TO (b4) de **STATUS** est mis à 0. En lisant ce bit en début de programme il est de plus possible de détecter si le PIC vient d'être mis sous tension ou si ce démarrage est dû à un débordement du WDT ("plantage" du programme ?).

c) Utilisation

Le WDT est donc une protection intéressante mais qu'il faut gérer par le programme. Pour cela il suffit de placer des instructions CLRWDT à intervalle inférieure au temps de débordement du WDT.

ΔT_{max} = rapport *prescaler* (d'après PS2 ,PS1, PS0) × 7 ms (cas le plus défavorable)

Il est à choisir en fonction de l'application selon le compromis :

ΔT_{max} faible (réaction rapide face au plantage) ↔ peu d'instructions CLRWDT à insérer

NB : - Le WD est mis en service par la directive CONFIG _WDT _ON à préciser au moment de la programmation (met le bit WDTE à 1 dans le registre CONFIG) et ne peut être désactivé par le programme.

- Ne jamais placer un CLRWDT dans une routine d'IT sinon l'IT risque de neutraliser la protection offerte par le WDT sans pour autant bien sûr réinitialiser le PIC.

8) Le mode SLEEP

L'instruction SLEEP permet de placer le PIC en mode SLEEP (Standby), c'est-à-dire que le PIC s'arrête juste après l'instruction en attendant d'être réveillé. L'intérêt de ce mode est de diminuer la consommation du PIC (Icc < 0,5 µA sans compter le courant fourni par chaque PORT en sortie).

Principe

Lorsque l'instruction SLEEP est exécutée :

- Le WDT est remis à 0, exactement comme le ferait une instruction CLRWDT.
- Le bit /TO du registre **STATUS** est mis à 1.
- Le bit /PD du registre **STATUS** est mis à 0.
- L'oscillateur est mis à l'arrêt, le PIC n'exécute plus aucune instruction.

Une fois dans cet état, le PIC est à l'arrêt. La consommation du circuit est réduite au minimum. Si le TIMER 0 est synchronisé par l'horloge interne, il est également mis dans l'incapacité de compter. Par contre, il est important de se rappeler que le *timer* du watchdog possède son propre circuit d'horloge; il continue donc de compter normalement.

Plusieurs événements peuvent faire sortir le PIC du mode SLEEP :

- Application d'un niveau 0 sur la broche /MCLR. Le PIC effectuera un reset classique à l'adresse 0x00. L'utilisateur pourra tester les bits /TO et /PD lors du démarrage pour vérifier l'événement concerné (reset, watch-dog, ou mise sous tension).

- Ecoulement du temps du *timer* du watchdog. Pour que cet événement réveille le PIC, il faut que le watchdog ait été mis en service dans les bits de configuration.
  Dans ce cas particulier, le débordement du WDT ne provoque pas un reset du PIC, il se contente de le réveiller. L'instruction qui suit est alors exécutée au réveil.

- Une interruption RB0/INT, RB, ou EEPROM est survenue. Pour qu'une telle interruption puisse réveiller le processeur, il faut que les bits de mise en service de l'interruption aient été positionnés. Le PIC se réveille et exécute l'instruction suivant SLEEP. On peut noter que si le bit GIE = 1 (interruptions activées) le PIC traite en plus l'IT qui l'a réveillé, sinon il continue.

9) Les accès en mémoire EEPROM

a) Ecriture lors de la programmation

Le PIC 16F84 dispose de 64 octets implantés à partir de l'adresse 0x2100 accessibles directement uniquement lors de la programmation.

b) Accès par le programme

L'EEPROM peut bien sûr être lue par le programme mais également être programmée grâce à des registres de contrôle :

- ***EEDATA*** *(0x08) :* C'est dans ce registre que va transiter la donnée à lire ou écrire .

- ***EEADR*** *(0x09):* Dans ce registre doit être précisée sur 8 bits l'adresse concernée par l'opération de lecture ou d'écriture en EEPROM. On voit déjà que pour cette famille de PICs, on ne peut pas dépasser 256 emplacements d'EEPROM. Pour le 16F84, la zone admissible va de 0x00 à 0x3F (codé en relatif par rapport à l'adresse de début de zone physique 0x2100), soit 64 emplacements.

EECON1 *(0x88) :* Ce registre contient 5 bits qui définissent ou indiquent le fonctionnement des cycles de lecture/écriture en EEPROM (voir datasheet)

EECON2 *(0x89):* Il s'agit tout simplement d'une adresse, qui sert à envoyer des commandes au PIC concernant les procédures d'écriture en EEPROM. On ne peut l'utiliser qu'en respectant la routine donnée par le constructeur (voir ②).

① Lecture

Pour lire une donnée en EEPROM, il suffit de placer l'adresse concernée dans le registre **EEADR**, puis de positionner le bit RD à 1. On peut ensuite récupérer la donnée lue dans le registre **EEDATA**.

② Ecriture

La procédure à suivre consiste d'abord à placer la donnée dans le registre EEDATA et l'adresse dans EEADR. Ensuite une séquence spécifique imposée par le constructeur doit être chargée dans le registre EECON2.

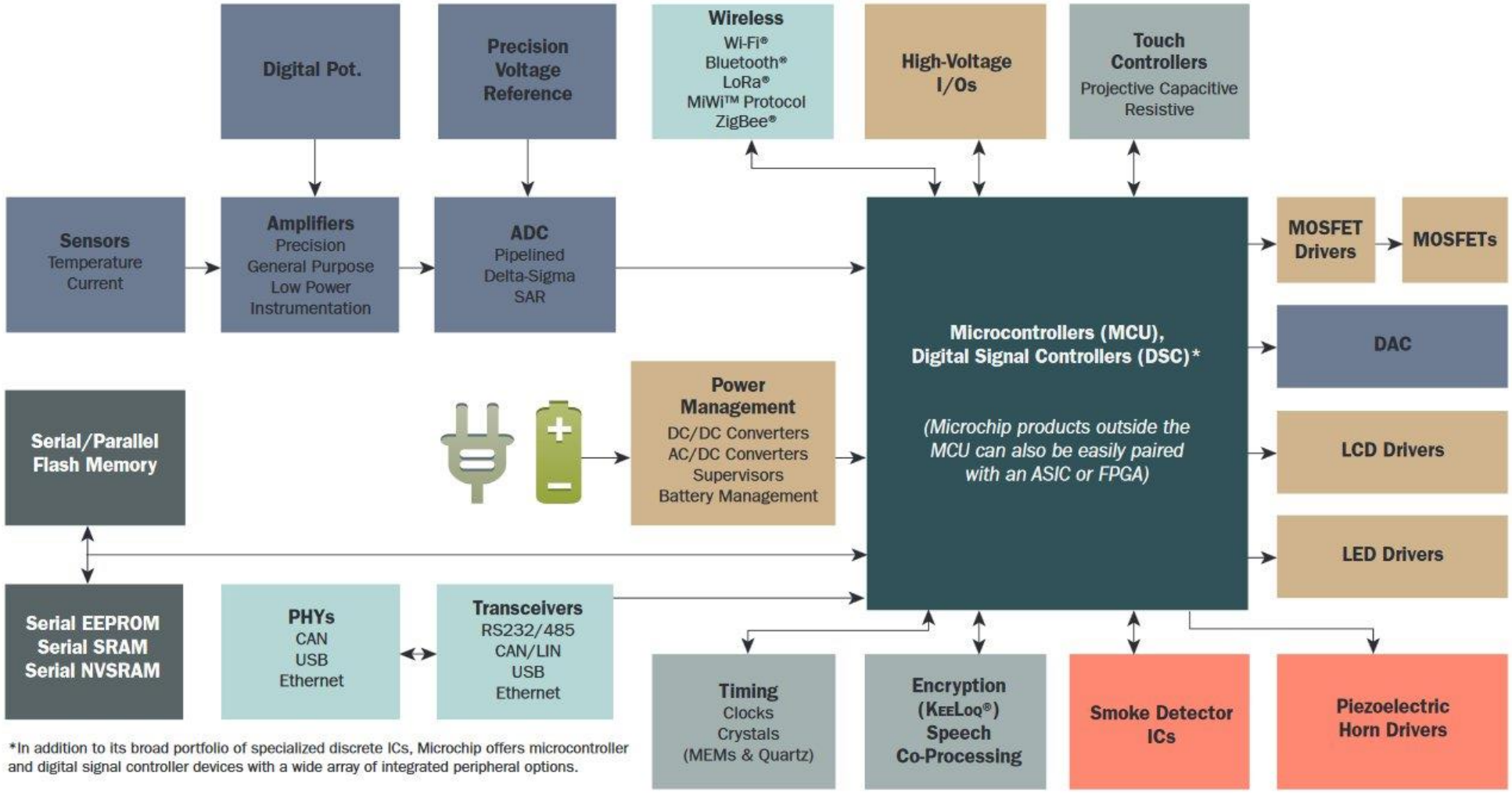
Remarques

- A la fin de la procédure d'écriture, la donnée n'est pas encore enregistrée dans l'EEPROM. Elle le sera approximativement 10 ms plus tard (ce qui représente tout de même environ 10.000 instructions !). On ne peut donc pas écrire une nouvelle valeur en EEPROM, ou lire cette valeur avant d'avoir vérifié la fin de l'écriture précédente.

- La fin de l'écriture peut être constatée par la génération d'une interruption (si le bit EEIE est positionné), ou par la lecture du flag EEIF (s'il avait été remis à 0 avant l'écriture), ou encore par consultation du bit WR qui est à 1 durant tout le cycle d'écriture.

NB : Avant d'écrire dans l'EEPROM il faut vérifier qu'une autre écriture n'est pas en cours en s'assurant que le bit WR du registre EECON1 est bien à 0.

Microchip Block Diagram Support



*In addition to its broad portfolio of specialized discrete ICs, Microchip offers microcontroller and digital signal controller devices with a wide array of integrated peripheral options.