

# Chapitre 6 : Programmation Assembleur LST GI

par  
Mohamed HASSOUN

# Éléments du chapitre

---

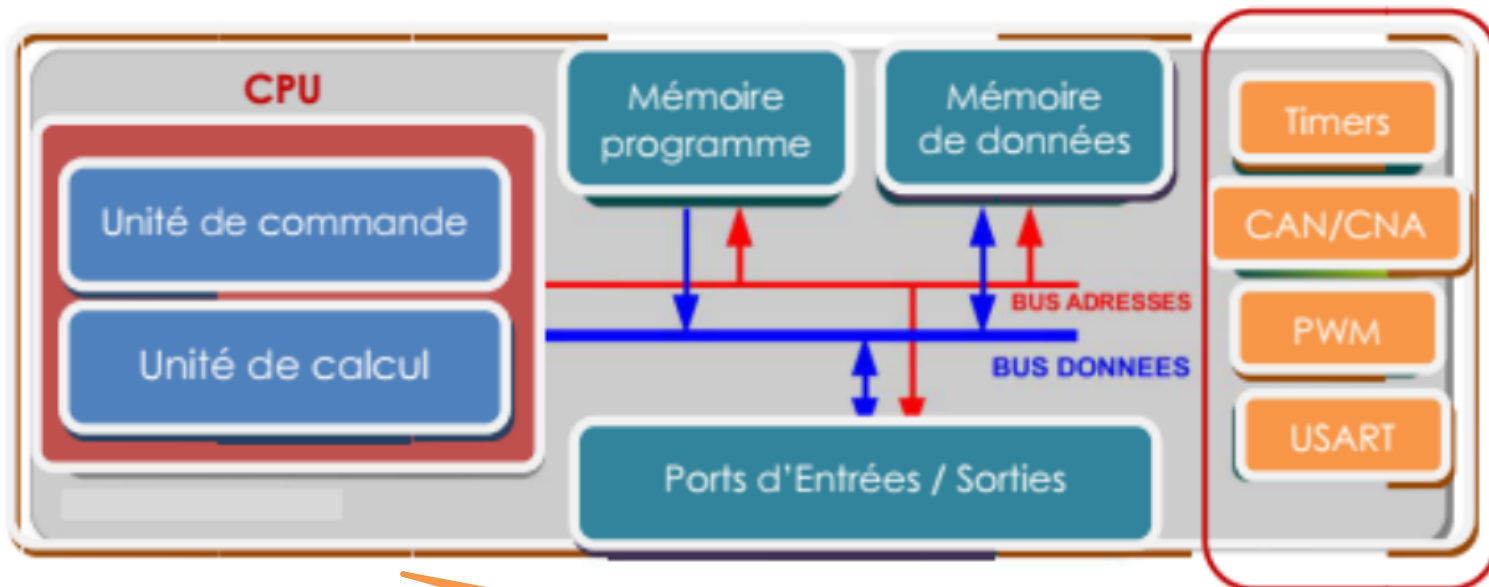
- Architecture interne du PIC
- Jeu d'instructions Assembleur
- Exemples d'implémentation

# Terminologie

## Le microcontrôleur Qu'est ce que c'est ?

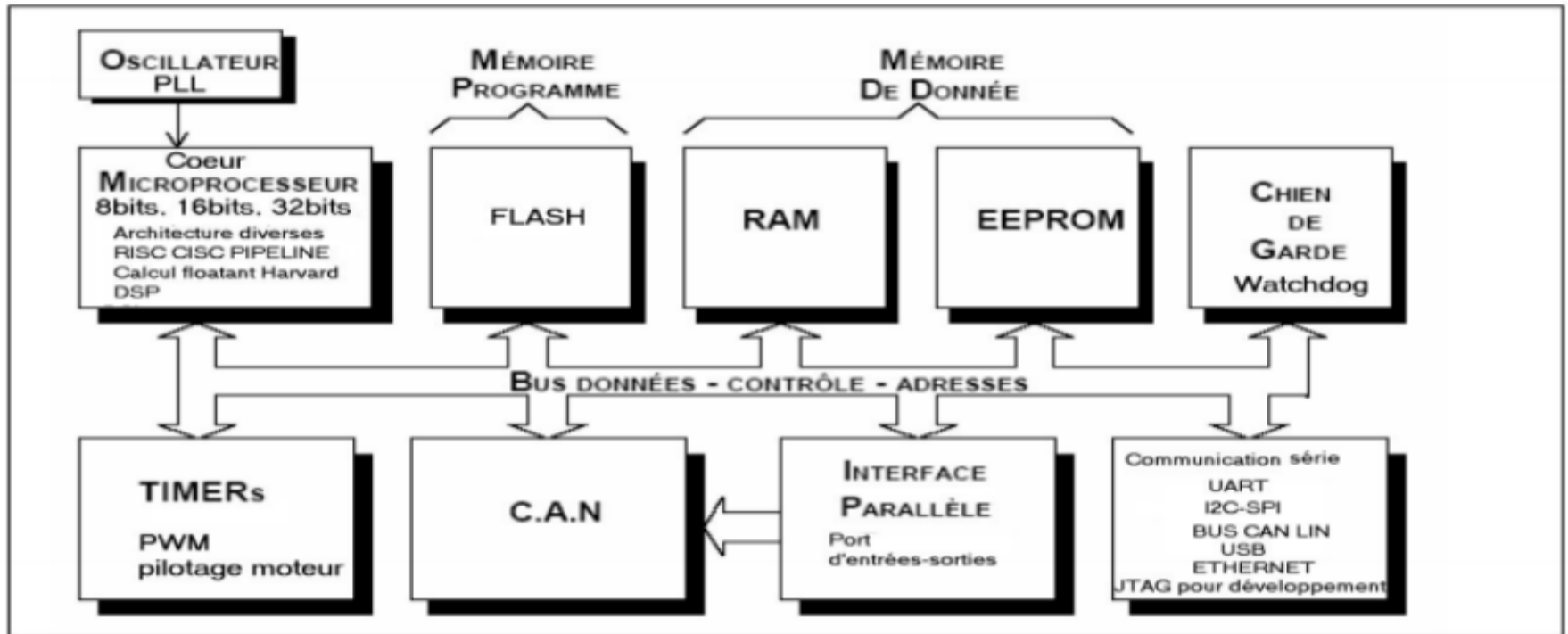
« Circuit intégré comprenant essentiellement un microprocesseur, des mémoires, et des éléments personnalisés selon l'application. »

Le microcontrôleur est un composant électronique programmable. On le programme par le biais d'un ordinateur grâce à un langage informatique, souvent propre au type de microcontrôleur utilisé






Il s'agit d'un micro ordinateur sur une seule puce

# Terminologie



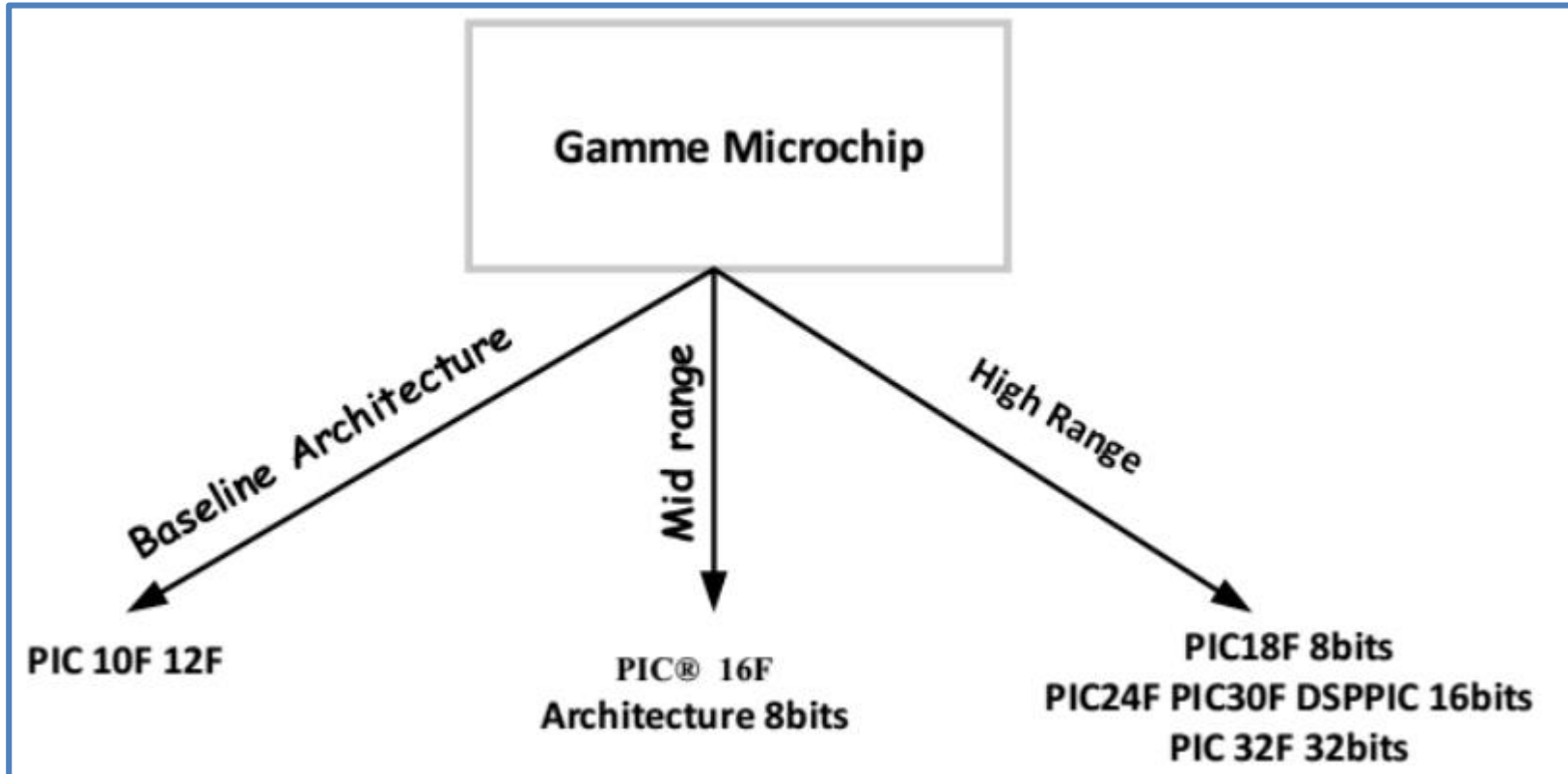
# Quelques fabricants

Fabricant	<i>Microchip</i>	<i>Atmel</i>	<i>Texas-Instrument</i>
Logo			
Famille	PIC et dsPIC	AVR, AVR32 et ARM	MSP430

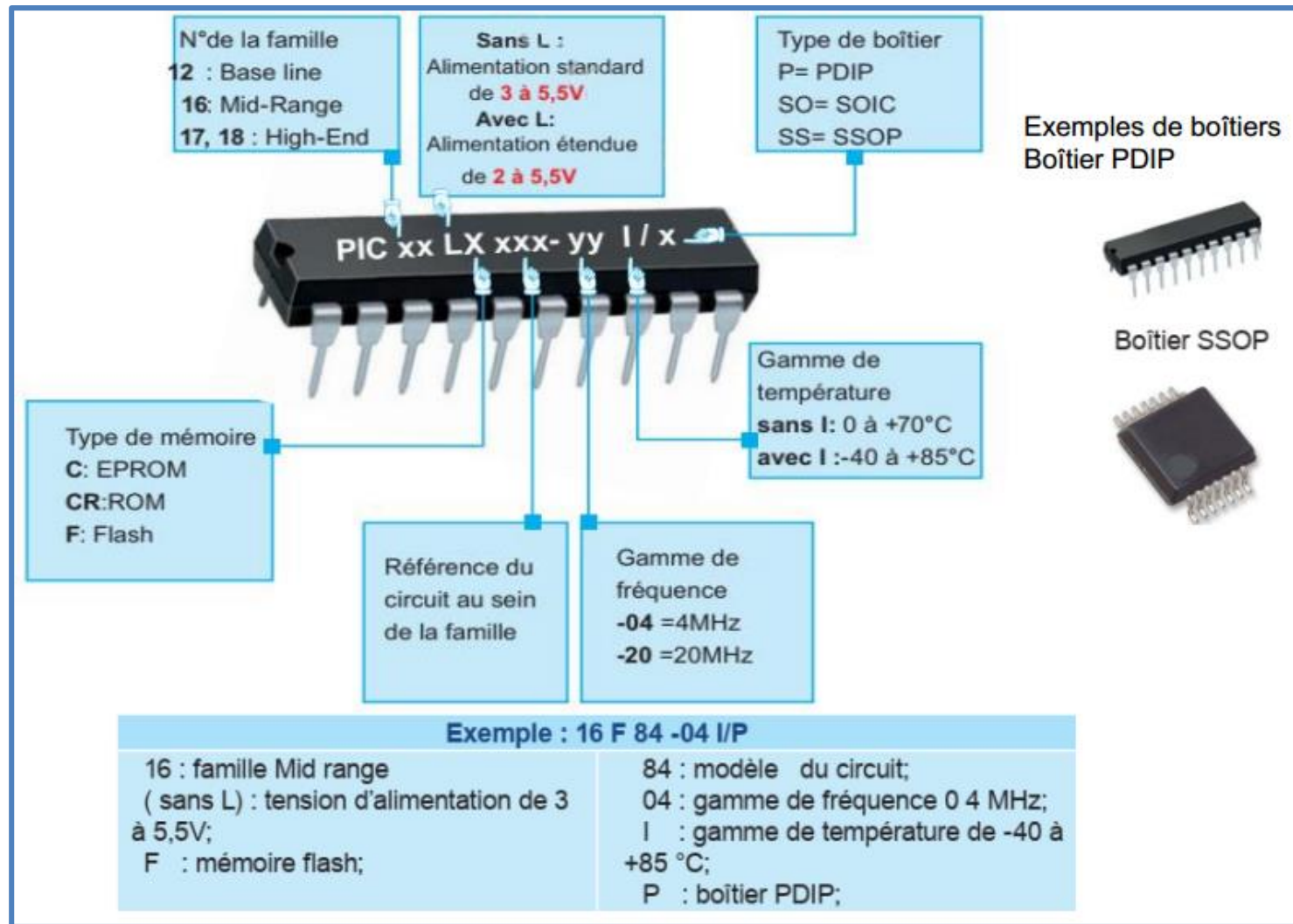
Analog devices	Fairchild	Temic	philips
Atmel	Infineon	Zilog	Motorola(freescale)
cypress	Maxim	Texas instrument	National semi
Dallas	Microchip	On semiconductor	St microelectronics

Avantages	Inconvénients
<ul style="list-style-type: none"> <li>• Mise en œuvre simple</li> <li>• Coûts de développement réduits</li> </ul>	<ul style="list-style-type: none"> <li>• Plus lent</li> <li>• Utilisation sous optimale</li> </ul>

# Gamme des MCU Microchip

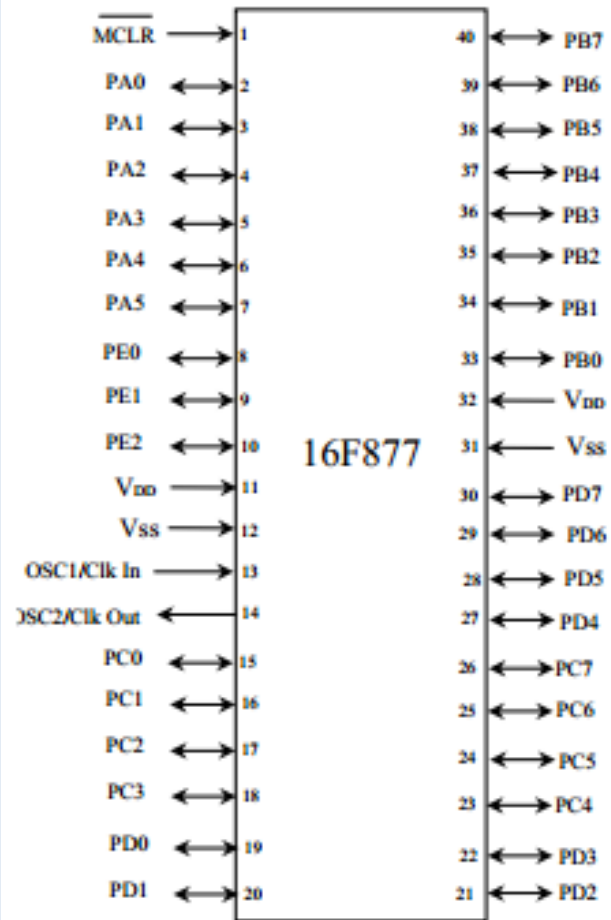


# Références



# Brochage du PIC16F877

## BROCHAGE : 16F877 ( 40 broches)





# Caractéristiques du PIC16F877

---

- Une mémoire programme de type EEPROM flash de 8K mots de 14 bits,
- Une RAM donnée de 368 octets,
- Une mémoire EEPROM de 256 octets,
- 05 ports d'entrée sortie, A (6 bits), B (8 bits), C (8 bits), D (8 bits) et E (3 bits)
- Convertisseur Analogiques numériques 10 bits à 8 entrées sélectionnables,
- USART, Port série universel, mode asynchrone (RS232) et mode synchrone
- SSP, Port série synchrone supportant I2C
- Trois TIMERS avec leurs Prescalers, TMR0, TMR1, TMR2
- Deux modules de comparaison et Capture CCP1 et CCP2
- 15 sources d'interruption,
- Générateur d'horloge, à quartz (jusqu'à 20 MHz)
- Tension de fonctionnement de 2 à 5V,
- Jeux de 35 instructions

# Caractéristiques du PIC16F877

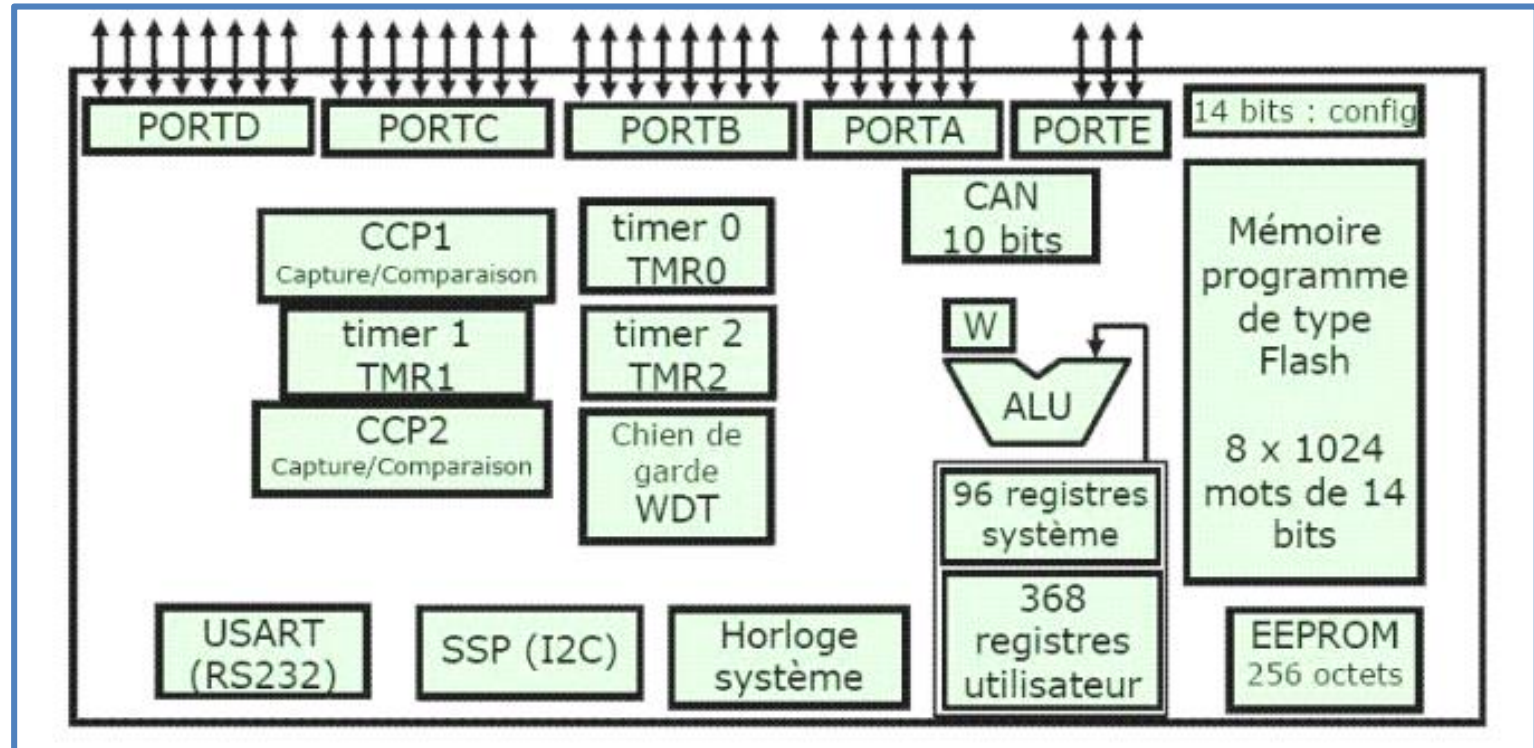
---

- 2 broches d'alimentation : VDD, VSS
- 2 broches d'horloge : OSC1, OSC2
- 1 broche de remise à zéro : MCLR
- 5 Ports d'interfaçage:
  - PORTA : sur 6 bits ( RA0 -> RA5)
  - PORTB : sur 8 bits (RB0 -> RB7)
  - PORTC : sur 8 bits (RC0 -> RC7)
  - PORTD : sur 8 bits (RD0 -> RD7)
  - PORTE : sur 3 bits (RE0 -> RE2)

## Architecture simplifiée du microcontrôleur PIC

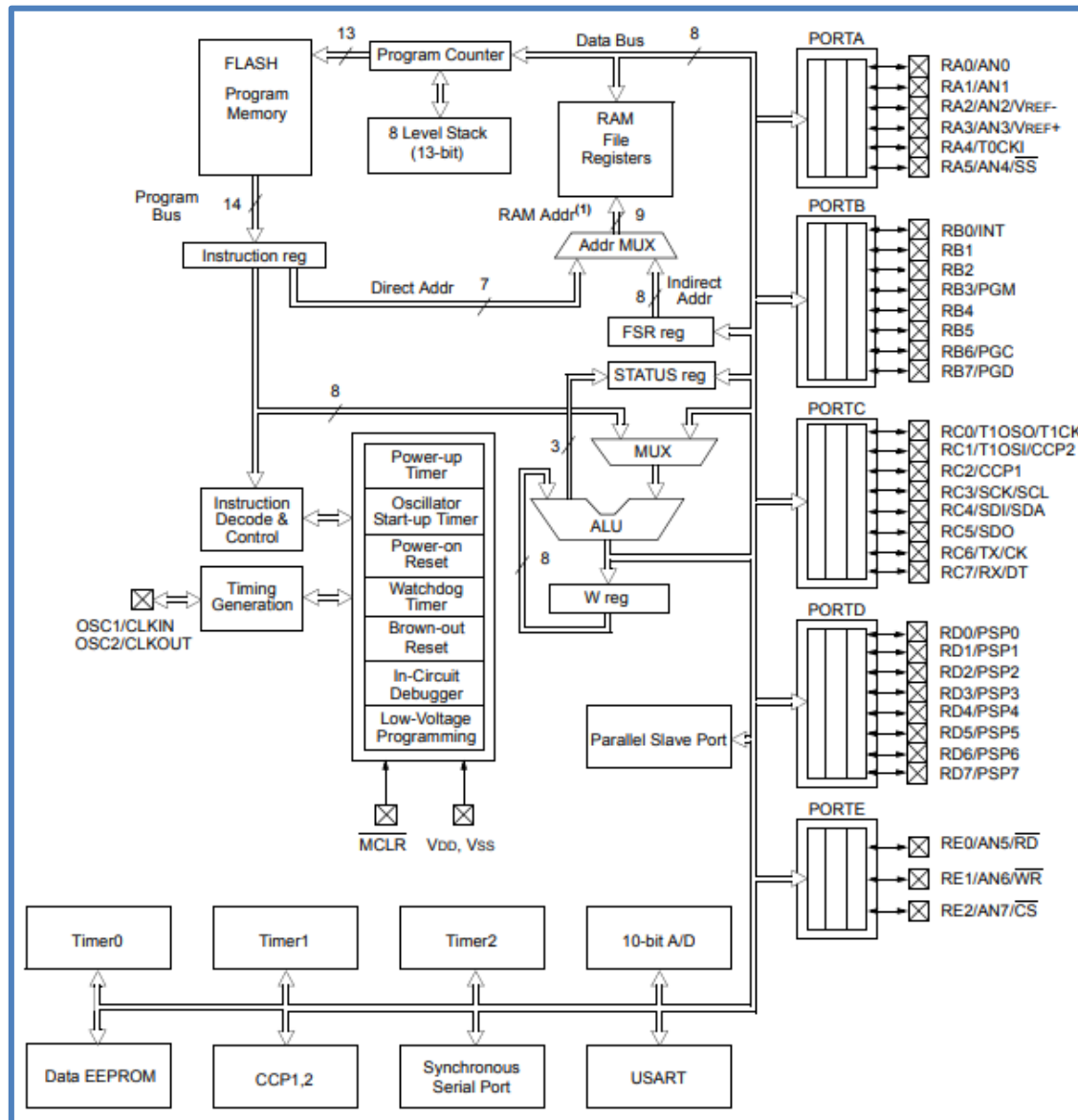


# Architecture Hardware



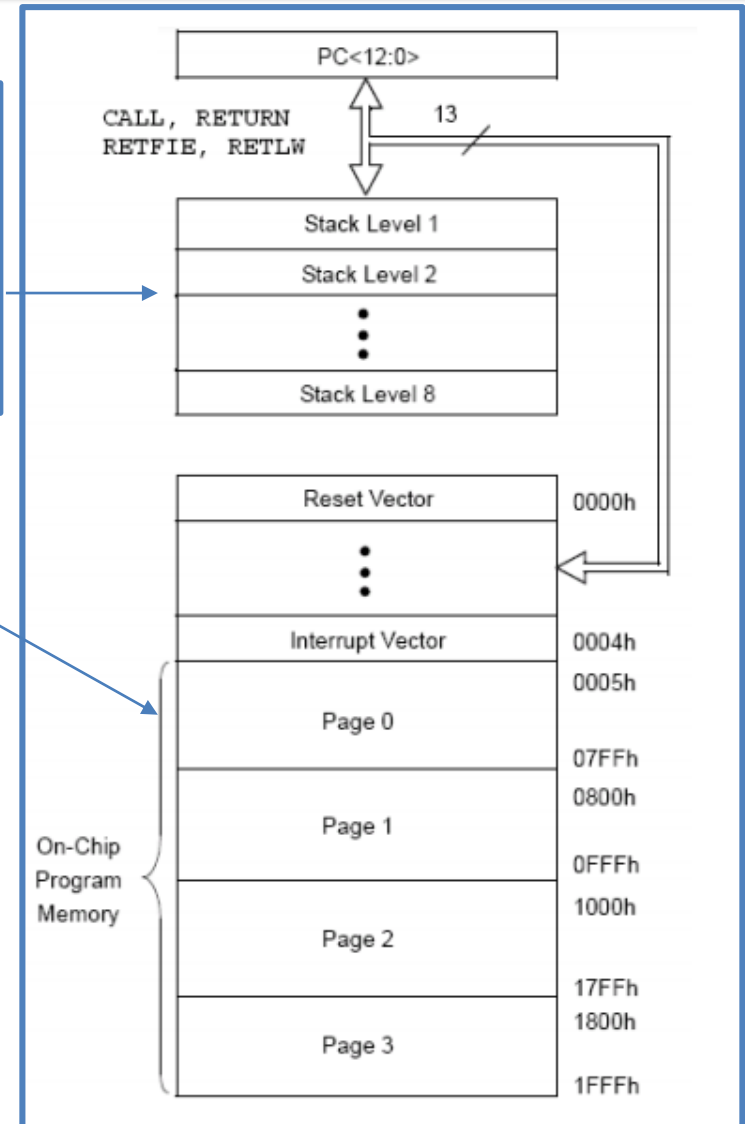
**Architecture Hardware simplifiée**

# Architecture Hardware



# Stack et EEPROM

- The PIC16F87XA family has an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. In the PIC microcontrollers, this is a special block of RAM memory used only for this purpose.
- Up to 8K x 14 words of FLASH Program Memory





# Cartographie de la mémoire volatile (RAM)

La mémoire RAM disponible du 16F877 est de 368 octets. Elle est répartie de la manière suivante :

- 96 octets en banque 0, adresses 0x20 à 0x7F
- 80 octets en banque 1, adresses 0xA0 à 0xEF
- 96 octets en banque 2, adresses 0x110 à 0x16F
- 96 octets en banque 3, adresses 0x190 à 0x1EF

## GPR (General Purpose Registers)

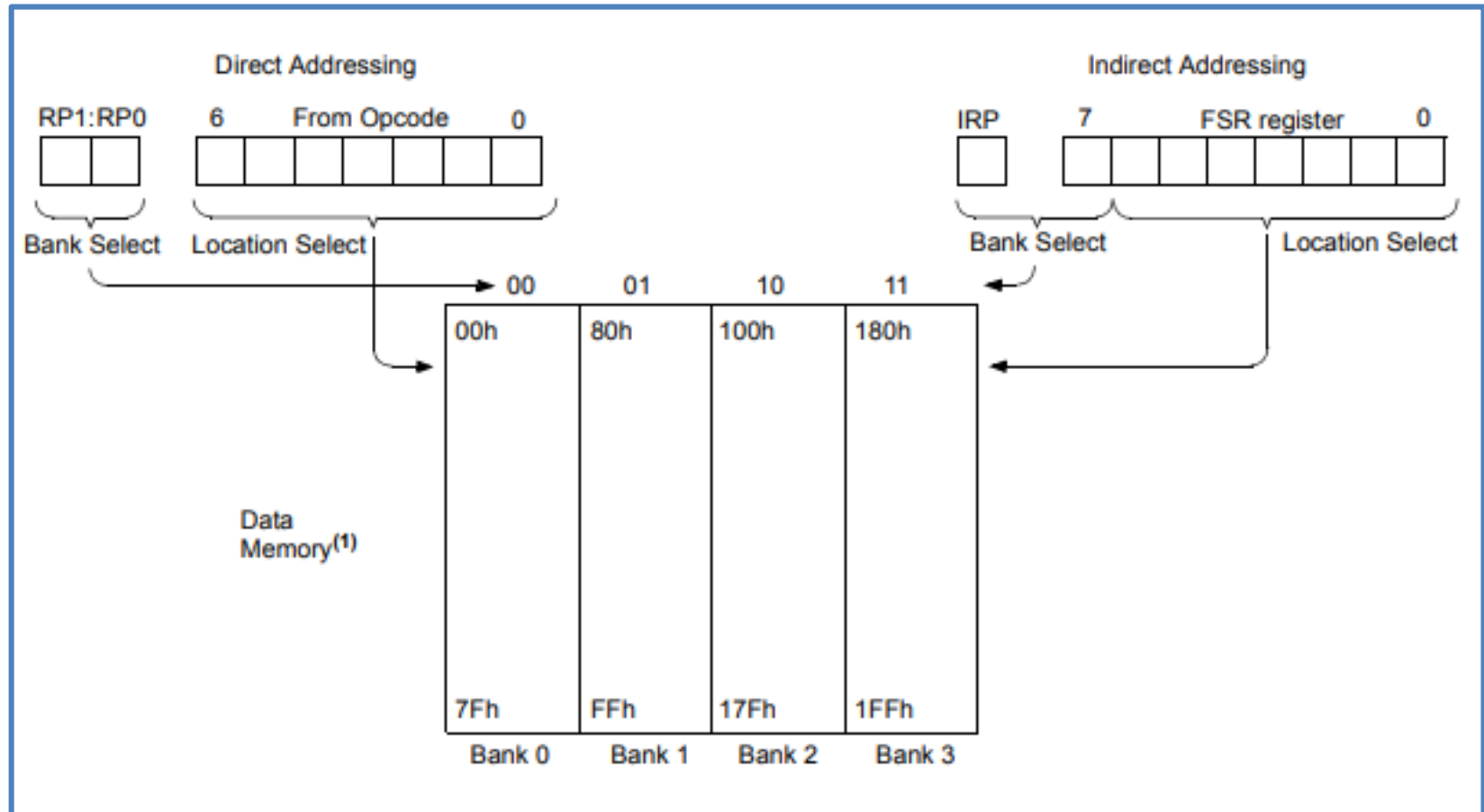
Espaces mémoires qui permet le stockage de données temporaires (variable, ...)

## SFR (Special Function Registers)

Registres de contrôle et d'état pour les périphériques

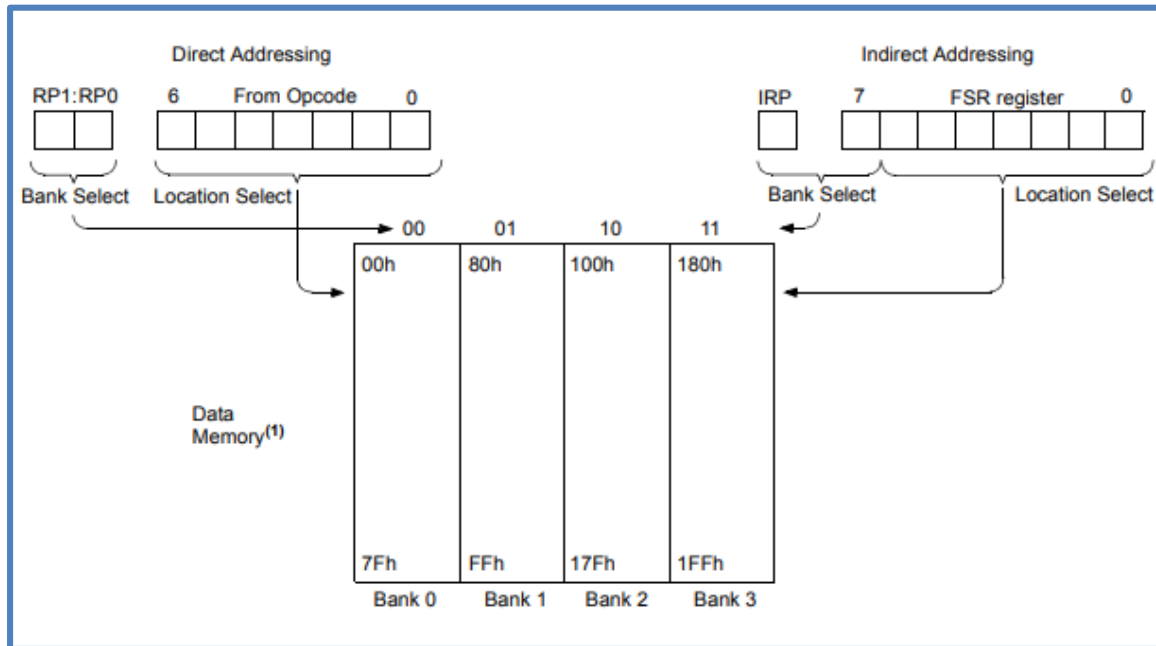
PAGE 0		PAGE 1		PAGE 2		PAGE 3	
00	INDF	80	INDF	100	INDF	180	INDF
01	TMR0	81	OPTION	101	TMR0	181	OPTION
02	PCL	82	PCL	102	PCL	182	PCL
03	STATUS	83	STATUS	103	STATUS	183	STATUS
04	FSR	84	FSR	104	FSR	184	FSR
05	PORTA	85	TRISA	105		185	
06	PORTB	86	TRISB	106	PORTB	186	TRISB
07	PORTC	87	TRISC	107		187	
08	PORTD(16F877)	88	TRISD(16F877)	108		188	
09	PORTE(16F877)	89	TRISE(16F877)	109		189	
0A	PCLATH	8A	PCLATH	10A	PCLATH	18A	PCLATH
0B	INTCON	8B	INTCON	10B	INTCON	18B	INTCON
0C	PIR1	8C	PIE1	10C	EEDATA	18C	EECON1
0D	PIR2	8D	PIE2	10D	EEADR	18D	EECON2
0E	TMR1L	8E	PCON	10E	EEDATH	18E	
0F	TMR1H	8F		10F	EEADRH	18F	
10	T1CON	90		110	RAM 16 octets	190	RAM 16 octets
11	TMR2	91	SSPCON2				
12	T2CON	92	PR2				
13	SSBUF	93	SSPADD				
14	SSPCON	94	SSPSTAT				
15	CCPR1L	95					
16	CCPR1H	96					
17	CCP1CON	97					
18	RCSTA	98	TXSTA				
19	TXREG	99	SPBRG				
1A	RCREG	9A					
1B	CCPR2L	9B					
1C	CCPR2H	9C					
1D	CCP2CONL	9D					
1E	ADRESH	9E	ADRESL				
1F	ADCON0	9F	ADCON1	11F		19F	
20	RAM 96 octets	A0	RAM 80 octets	120	RAM 80 octets	1A0	RAM 80 octets
7F		EF		16F		1EF	
		F0		170		1F0	
		FF		17F		1FF	

# Méthodes d'accès mémoire





# Méthodes d'accès mémoire



Example of direct addressing:

1. TEMP Equ 0x030
2. Movlw 5
3. Movwf TEMP

Example of indirect addressing:

1. TEMP Equ 0x030
2. Movlw 0x030
3. Movwf FSR
4. Movlw 5
5. Movwf INDF

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

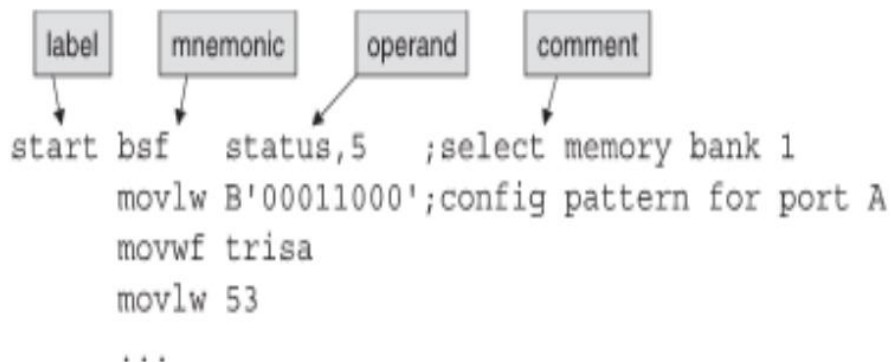
# Structure d'un code Assembleur

Un programme en assembleur comporte une *instruction* par ligne.

Une ligne se découpe en quatre colonnes

1. la 1ère colonne commence en tout début de ligne et contient au plus un mot appelé symbole ou étiquette associé à l'adresse de la case mémoire de l'instruction ou de la donnée courante. On peut aussi attribuer une valeur quelconque à une étiquette.
2. la 2ème colonne commence après un espace ou une tabulation. elle contient une instruction, une macro-instruction ou une directive.
3. la 3ème colonne est séparée par un espace ou une tabulation contient les arguments séparés par des virgules de la 2ème colonne.
4. la 4ème colonne commence par un ; s'achève au retour chariot contient un commentaire.

## Exemple :



```
start  bsf    status,5    ;select memory bank 1
      movlw  B'00011000';config pattern for port A
      movwf  trisa
      movlw  53
      ...
```

# Jeu d'instructions Assembleur

{W,F ? d} signifie que le résultat va soit dans W si d=0 ou w, soit dans F si d= 1 ou f

INSTRUCTIONS OPERANT SUR REGISTRE			indicateurs	Cycles
<b>ADDWF</b>	<b>F,d</b>	$W + F \rightarrow \{W, F ? d\}$	C,DC,Z	1
<b>ANDWF</b>	<b>F,d</b>	$W \text{ and } F \rightarrow \{W, F ? d\}$	Z	1
<b>CLRF</b>	<b>F</b>	Clear F	Z	1
<b>COMF</b>	<b>F,d</b>	Complémente F $\rightarrow \{W, F ? d\}$	Z	1
<b>DECf</b>	<b>F,d</b>	décrémente F $\rightarrow \{W, F ? d\}$	Z	1
<b>DECFSZ</b>	<b>F,d</b>	décrémente F $\rightarrow \{W, F ? d\}$ skip if 0		1(2)
<b>INCF</b>	<b>F,d</b>	incrémente F $\rightarrow \{W, F ? d\}$	Z	1
<b>INCFSZ</b>	<b>F,d</b>	incrémente F $\rightarrow \{W, F ? d\}$ skip if 0		1(2)
<b>IORWF</b>	<b>F,d</b>	$W \text{ or } F \rightarrow \{W, F ? d\}$	Z	1
<b>MOVF</b>	<b>F,d</b>	$F \rightarrow \{W, F ? d\}$	Z	1
<b>MOVWF</b>	<b>F</b>	$W \rightarrow F$		1
<b>RLF</b>	<b>F,d</b>	rotation à gauche de F a travers C $\rightarrow \{W, F ? d\}$	C	1
<b>RRF</b>	<b>F,d</b>	rotation à droite de F a travers C $\rightarrow \{W, F ? d\}$		1
<b>SUBWF</b>	<b>F,d</b>	$F - W \rightarrow \{W, F ? d\}$	C,DC,Z	1
<b>SWAPF</b>	<b>F,d</b>	permuté les 2 quartets de F $\rightarrow \{W, F ? d\}$		1
<b>XORWF</b>	<b>F,d</b>	$W \text{ xor } F \rightarrow \{W, F ? d\}$	Z	1

INSTRUCTIONS OPERANT SUR BIT				
<b>BCF</b>	<b>F,b</b>	RAZ du bit b du registre F		1
<b>BSF</b>	<b>F,b</b>	RAU du bit b du registre F		1
<b>BTFSC</b>	<b>F,b</b>	teste le bit b de F, si 0 saute une instruction		1(2)
<b>BTFSS</b>	<b>F,b</b>	teste le bit b de F, si 1 saute une instruction		1(2)

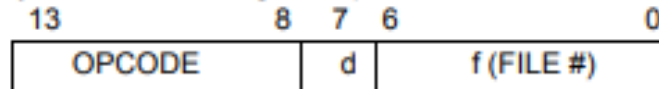
INSTRUCTIONS OPERANT SUR CONSTANCE				
<b>ADDLW</b>	<b>K</b>	$W + K \rightarrow W$	C,DC,Z	1
<b>ANDLW</b>	<b>K</b>	$W \text{ and } K \rightarrow W$	Z	1
<b>IORLW</b>	<b>K</b>	$W \text{ or } K \rightarrow W$	Z	1
<b>MOVLW</b>	<b>K</b>	$K \rightarrow W$		1
<b>SUBLW</b>	<b>K</b>	$K - W \rightarrow W$	C,DC,Z	1
<b>XORLW</b>	<b>K</b>	$W \text{ xor } K \rightarrow W$	Z	1

# Jeu d'instructions Assembleur

AUTRES INSTRUCTIONS			
<b>CLRW</b>	Clear W	Z	1
<b>CLRWD</b>	Clear Watchdog timer	TO', PD'	1
<b>CALL</b> <b>L</b>	Branchement à un sous programme de label L		2
<b>GOTO</b> <b>L</b>	branchement à la ligne de label L		2
<b>NOP</b>	No operation		1
<b>RETURN</b>	retourne d'un sous programme		2
<b>RETFIE</b>	Retour d'interruption		2
<b>RETLW</b> <b>K</b>	retourne d'un sous programme avec K dans W		2
<b>SLEEP</b>	se met en mode standby	TO', PD'	1

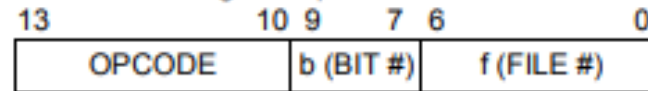
# Jeu d'instructions Assembleur

## Byte-oriented file register operations



d = 0 for destination W  
d = 1 for destination f  
f = 7-bit file register address

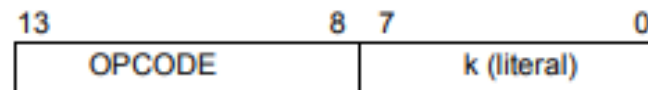
## Bit-oriented file register operations



b = 3-bit bit address  
f = 7-bit file register address

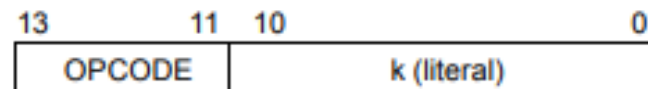
## Literal and control operations

### General



k = 8-bit immediate value

### CALL and GOTO instructions only



k = 11-bit immediate value

# Jeu d'instructions Assembleur

**ADDWF 70h,1**                      ou                      **ADDWF 70h,f**

Signifie : additionner le contenu de W avec le contenu de la case mémoire d'adresse 70h et placer le résultat dans la case mémoire 70h

**XORWF 35h,0**                      ou                      **XORWF 35h,w**

Signifie : faire un ou exclusif entre W et le contenu de la case mémoire d'adresse 35h et placer le résultat dans l'accumulateur W

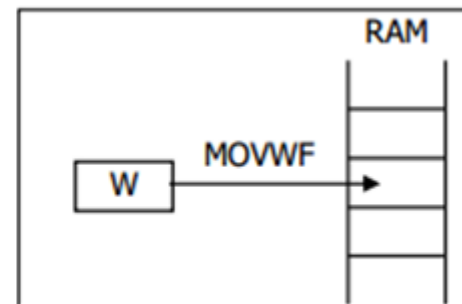
**BSF STATUS,2**                      ; signifie : placer à 1 le bit 2 (3ème bit à partir de la droite) du registre STATUS

**BCF 45h,6**                      ; signifie : placer à 0 le bit 6 (7ème bit à partir de la droite) du registre de la case mémoire d'adresse 45h

**MOVWF** permet de copier l'accumulateur W dans un registre (SFR ou GPR):

**MOVWF STATUS**                      ; signifie : Copier le contenu de W dans le registre STATUS

**MOVWF 55h**                      ; signifie : Copier le contenu de W dans la case mémoire d'adresse 55h



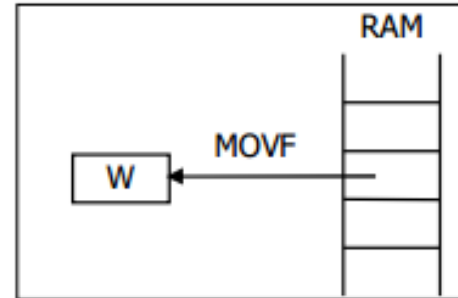
# Jeu d'instructions Assembleur

**MOVF STATUS,0** ; Copier le contenu du registre **STATUS** dans l'accumulateur **W**

**MOVF 35h,0** ; Copier le contenu de la case mémoire d'adresse **35h** dans l'accumulateur **W**

Avec le paramètre  $d=1$ , l'instruction **MOVF** semble inutile car elle permet de copier un registre sur lui-même ce qui à priori ne sert à rien.

**MOVF STATUS,1** ; Copier le contenu du registre **STATUS** dans lui même



**btfsc F,b** : **bit test skip if clear** : teste le bit **b** du registre **F** et saute l'instruction suivante si le bit testé est nul

**btfss F,b** : **bit test skip if set** : teste le bit **b** du registre **F** et saute l'instruction suivante si le bit testé est égal à 1

**exemple :**

```
sublw    100      ; 100 - W → W
btfss    STATUS,Z  ; tester le bit Z du registre STATUS et sauter une ligne si Z=1
clrf     70h      ; après btfss, le programme continue ici si Z=0
cmpf     70h,f    ; après btfss, le programme continue ici si Z=1
suite du programme
suite du programme
...
```

# Jeu d'instructions Assembleur

---

***Incfsz F,1*** : **increment skip if Z** : incrémente le registre F et sauter une ligne si le résultat = 0. Le paramètre 1 indique que le résultat de l'incrémentement doit aller dans F.

***deccfsz F,1*** : **decrement skip if Z** : décrémente le registre F et sauter une ligne si le résultat = 0. Le paramètre 1 indique que le résultat de la décrémentation doit aller dans F.

*Instruction 1*  
*Instruction 2*  
***Goto*** ***bonjour***  
*instruction 3*  
*instruction 4*  
*instruction 5*  
*instruction 6*  
*instruction 7*

***bonjour***

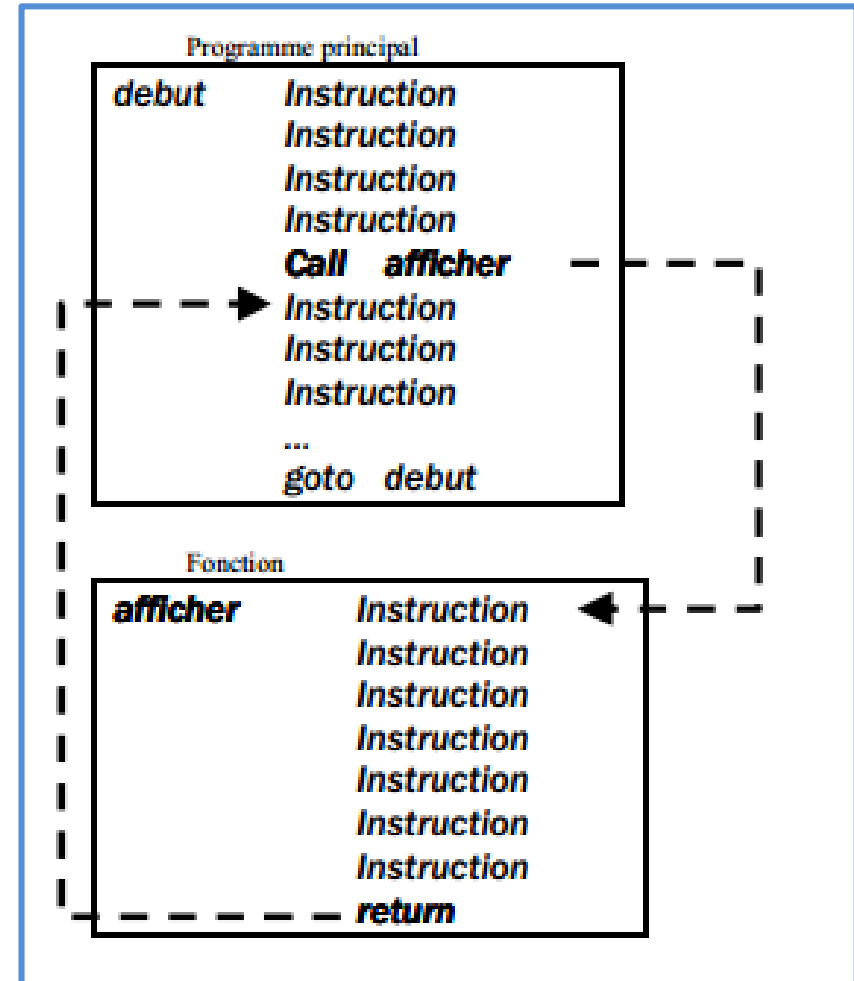


# Jeu d'instructions Assembleur

The CALL instruction is used to jump to a subroutine, which must be terminated with the RETURN instruction. CALL has the address of the first instruction in the subroutine as its operand. When the CALL instruction is executed, the destination address is copied to the PC.

The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POP'ed in the event of a RETURN, RETLW or a RETFIE instruction execution.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).



# Bits d'états

**Z** : passe à 1 quand le résultat d'une instruction est nul

**C** : passe à 1 quand l'opération a généré une retenue

**DC** : passe à 1 quand les 4<sup>ème</sup> bits génère une retenue

STATUS	IRP	RP1	RP0			Z	DC	C
--------	-----	-----	-----	--	--	---	----	---

$F - W = 0 \implies Z=1, C=1, B=0 \implies$  pas de retenue de soustraction

$F - W > 0 \implies Z=0, C=1, B=0 \implies$  pas de retenue de soustraction

$F - W < 0 \implies Z=0, C=0, B=1 \implies$  il ya retenue de soustraction

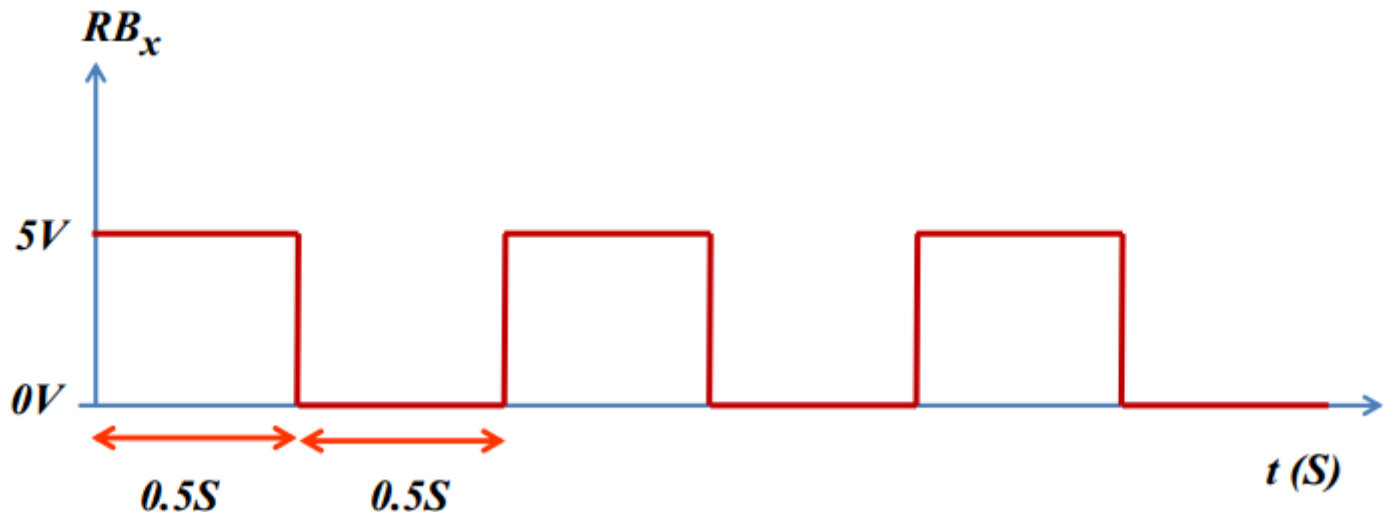
Pour réaliser une comparaison entre F et W, on fait  $F - W$  et on observe Z et C

- $Z=1 \implies$  égalité
- $C=1 \implies$  F sup ou égal à W
- $C=0 \implies$  F inférieur à W

# Exemple d'implémentation

## Exercice:

*Soit un PIC cadencé par un quartz de fréquence 4MHz qui exécute un cycle d'instruction par 1 $\mu$ s, proposer un programme, en assembleur qui génère sur une des broches PORTB un signal carré de fréquence  $f=1\text{Hz}$ .*



# Exemple d'implémentation

---

- Temporisation avec une boucle

```
aa      movlw M  
        movwf 0x20  
        decfsz 0x20,f  
        goto aa
```

$$T = (3M+1)T_{cyc}$$

# Exemple d'implémentation

---

- Temporisation avec deux boucles imbriquées

```
        movlw N  
        movwf 0x20  
cc      movlw M  
        movwf 0x21  
dd      decfsz 0x21,f  
        goto dd  
        decfsz 0x20,f  
        goto cc
```

$$T \sim (3NM+3)T_{cyc}$$

# Exemple d'implémentation

- Temporisation avec trois boucles imbriquées

```
        movlw N
        movwf 0x20
aa      movlw M
        movwf 0x21
cc      movlw K
        movwf 0x22
dd      decfsz 0x22,f
        goto dd
        decfsz 0x21,f
        goto cc
        decfsz 0x20,f
        goto aa
```

$$T \sim (1+3MNK-4MN-3MK+M)T_{cyc} \sim 3MNK.T_{cyc}$$

# Solution

```
list p=16f877
include <p16f877.inc>

bcf STATUS,RP1
bsf STATUS,RP0
bcf TRISB,0
bcf STATUS,RP0

start
    bsf PORTB,0 ; allumer la LED
    call tempo ; appeler la tempo de 0.5s
    bcf PORTB,0 ; éteindre LED
    call tempo ; appeler la tempo de 0.5s
    goto start ; boucler

tempo
    movlw 0x37
    movwf 0x20
aa    movlw 0x37
    movwf 0x21
cc    movlw 0x37
    movwf 0x22
dd    decfsz 0x22,f
    goto dd
    decfsz 0x21,f
    goto cc
    decfsz 0x20,f
    goto aa
    return

end
```