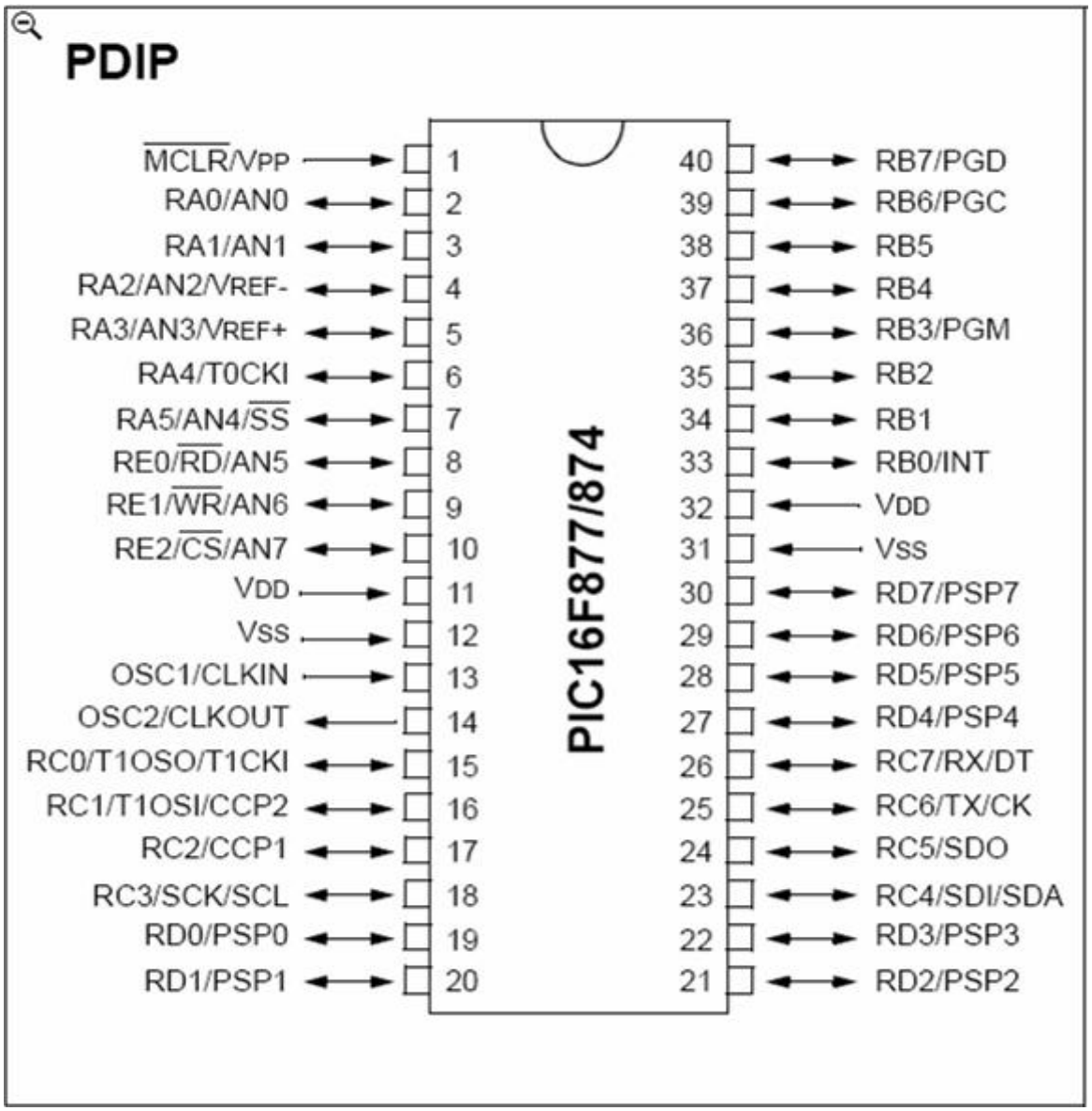
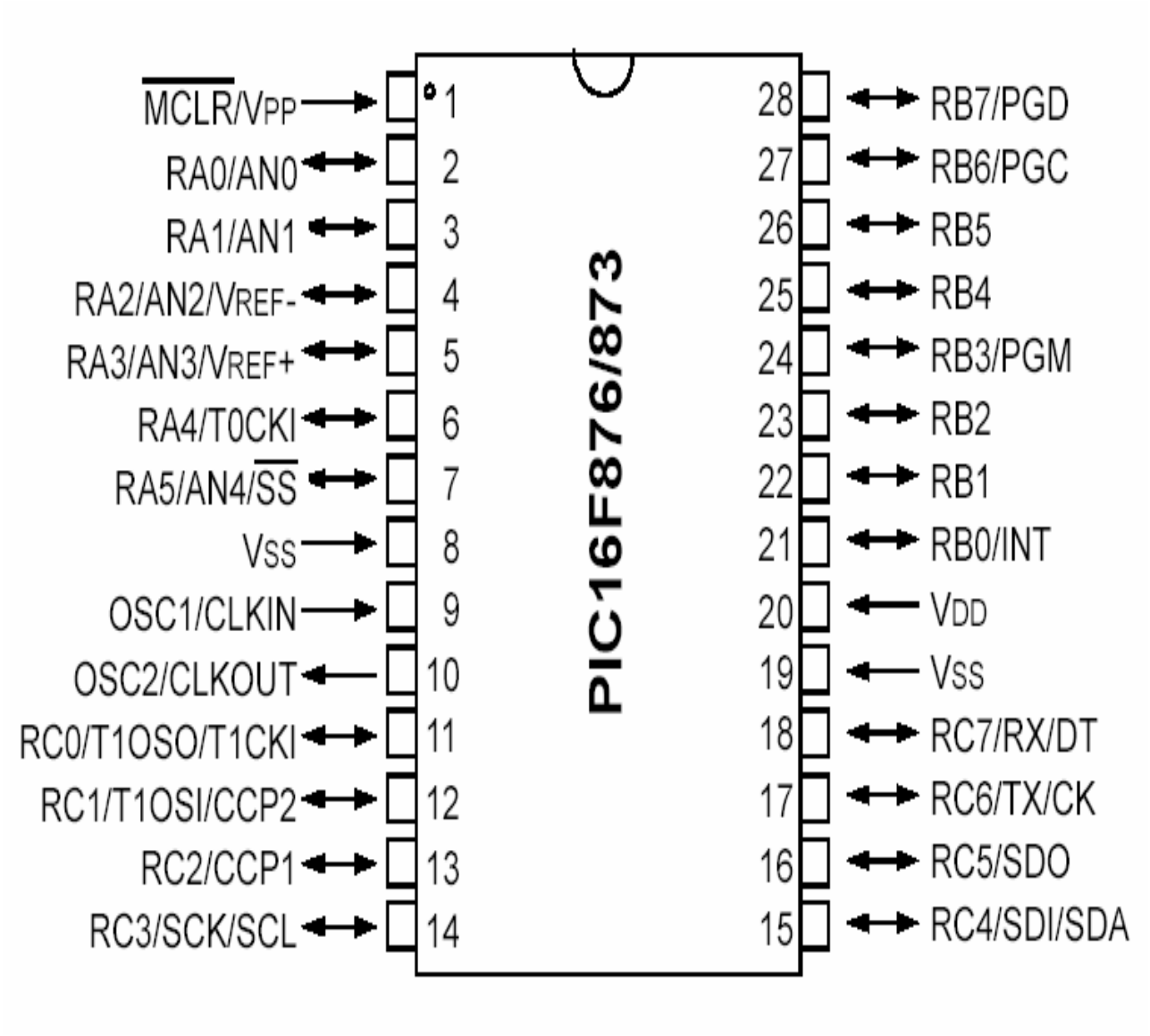
MICROCONTROLEURS

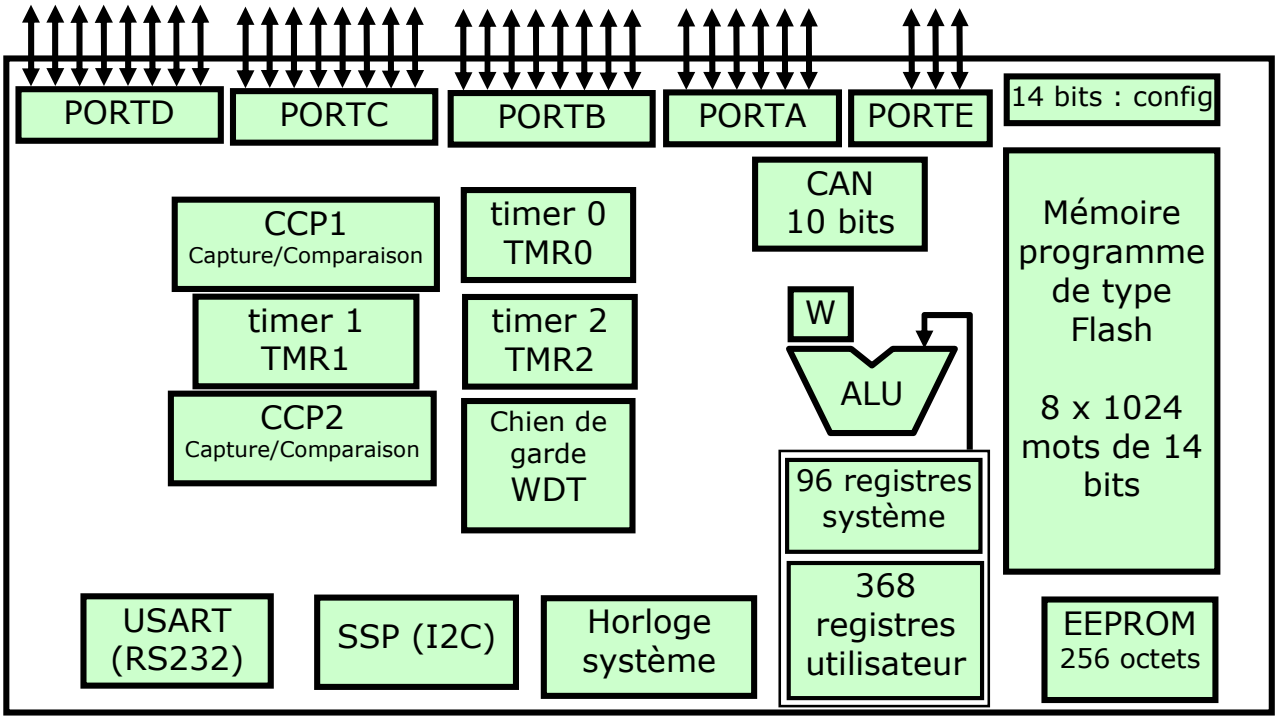
Famille Mid-Range
de Microchip

LE PIC 16F876/877

A. Oumnad




PIC16F876/873
MCLR/VPP 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF- 4
RA3/AN3/VREF+ 5
RA4/T0CKI 6
RA5/AN4/SS 7
VSS 8
OSC1/CLKIN 9
OSC2/CLKOUT 10
RC0/T1OSO/T1CKI 11
RC1/T1OSI/CCP2 12
RC2/CCP1 13
RC3/SCK/SCL 14
28 RB7/PGD
27 RB6/PGC
26 RB5
25 RB4
24 RB3/PGM
23 RB2
22 RB1
21 RB0/INT
20 VDD
19 VSS
18 RC7/RX/DT
17 RC6/TX/CK
16 RC5/SDO
15 RC4/SDI/SDA


PORTD
PORTC
PORTB
PORTA
PORTE
14 bits : config
CCP1
Capture/Comparaison
timer 0
TMR0
CAN
10 bits
Mémoire programme de type Flash
8 x 1024 mots de 14 bits
timer 1
TMR1
timer 2
TMR2
W
ALU
CCP2
Capture/Comparaison
Chien de garde WDT
96 registres système
368 registres utilisateur
USART (RS232)
SSP (I2C)
Horloge système
EEPROM
256 octets

SOMMAIRE

I Introduction

I.1 Les microcontrôleurs en général

Un microcontrôleur est un composant électronique *Autonome* doté :

- d'un microprocesseur,
- de la mémoire RAM,
- de la mémoire permanente
- des interfaces d'E/S //, série (RS232,I2C …)
- des interfaces d'E/S analogique
- Des Timer pour gérer le temps
- D'autres module plus au moins sophistiqués selon la taille des µC

Il est généralement moins puissant qu'un microprocesseur en terme de rapidité ou de taille mémoire, il se contente le plus souvent d'un bus 8 ou 16 bits. Ceci en fait un composant **très bon marché** parfaitement Adapté pour piloter les applications embarquées dans de nombreux domaines d'application. Je pense qu'on ne se tromperait pas beaucoup si on affirme qu'aujourd'hui il y'a un microcontrôleur (± grand) dans chaque équipement électronique :

- Informatique (souris, modem …)
- Vidéo (Appareil photos numérique, caméra numérique …)
- Contrôle des processus industriels (régulation, pilotage)
- Appareil de mesure (affichage, calcul statistique, mémorisation)
- Automobile (ABS, injection, GPS, airbag)
- Multimédia (téléviseur,carte audio, carte vidéo, MP3, magnétoscope)
- Téléphones (fax, portable, modem)
- Electroménager (lave-vaisselle, lave-linge, four micro-onde)

Un microcontrôleur peut être programmé une fois pour toutes afin qu'il effectue une ou des tâches précises au sein d'un appareil électronique. Mais les µC récents peuvent être reprogrammés et ceci grâce à leur mémoire permanente de type FLASH (d'où le terme flasher quelque chose)

Plusieurs Constructeurs se partagent le marché des microcontrôleurs, citons INTEL, MOTOROLA, AMTEL, ZILOG, PHILIPS et enfin MICROCHIP avec ses PICs très populaires qui nous intéresse ici dans ce cours.

Les microcontrôleurs, quelque soit leurs constructeurs, ont des architecture très similaires et sont constitués de modules fondamentaux assurant les mêmes fonctions : UAL, Ports d'E/S, interfaces de communications série, Interfaces d'E/S analogiques, Timers et horloge temps réels …On peut dire que seul le langage de programmation (Assembleurs) constitue la différence

majeure en deux microcontrôleur (similaires) venant de deux constructeurs différents.

Nous avons choisit dans ce cours d'apprendre les microcontrôleurs à travers une étude détaillée des microcontrôleur 16F87x (x=3, 4, 6, 7) qui constitue les éléments fondamentaux de la famille mid-range qui est la famille « moyenne puissance » de Microchip

I.2 Les PICs de Microchip

Les PICs sont des microcontrôleurs à architecture RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide ce qui augmente la vitesse de fonctionnement du microcontrôleur.

La famille des PICs est subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits, la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie la 16F84 et 16F876), et la famille **High-End**, qui utilise des mots de 16 bits.

Les PICs sont des composants STATIQUES, Ils peuvent fonctionner avec des fréquences d'horloge allant du continu jusqu'à une fréquence max spécifique à chaque circuit. Un PIC16F876-04 peut fonctionner avec une horloge allant du continu jusqu'à 4 MHz.

Nous nous limiterons dans ce document à la famille Mid-Range et particulièrement au PIC 16F876/877, sachant que si on a tout assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur.

PIC	FLASH	RAM	EEPROM	I/O	A/D	Port //	Port Série
16F870	2K	128	64	22	5	NON	USART
16F871	2K	128	64	33	8	PSP	USART
16F872	2K	128	64	22	5	NON	MSSP
16F873	4K	192	128	22	5	NON	USART/MSSP
16F874	4K	192	128	33	8	PSP	USART/MSSP
16F876	8K	368	256	22	5	NON	USART/MSSP
16F877	8K	368	256	33	8	PSP	USART/MSSP

Tableau I.1 : différents circuit de la famille 16F87X

Les éléments essentiels du PIC 16F876 sont :

- 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,
- Trois ports d'entrée sortie, A (6 bits), B (8 bits), C (8 bits),
- Convertisseur Analogiques numériques 10 bits à 5 canaux,
- 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
- Un chien de garde,
- 13 sources d'interruption,
- Générateur d'horloge, à quartz (jusqu' à 20 MHz) ou à Oscillateur RC
- Protection de code,
- Fonctionnement en mode sleep pour réduction de la consommation,
- Programmation par mode ICSP (*In Circuit Serial Programming*) 12V ou 5V,
- Possibilité aux applications utilisateur d'accéder à la mémoire programme,
- Tension de fonctionnement de 2 à 5V,
- Jeux de 35 instructions

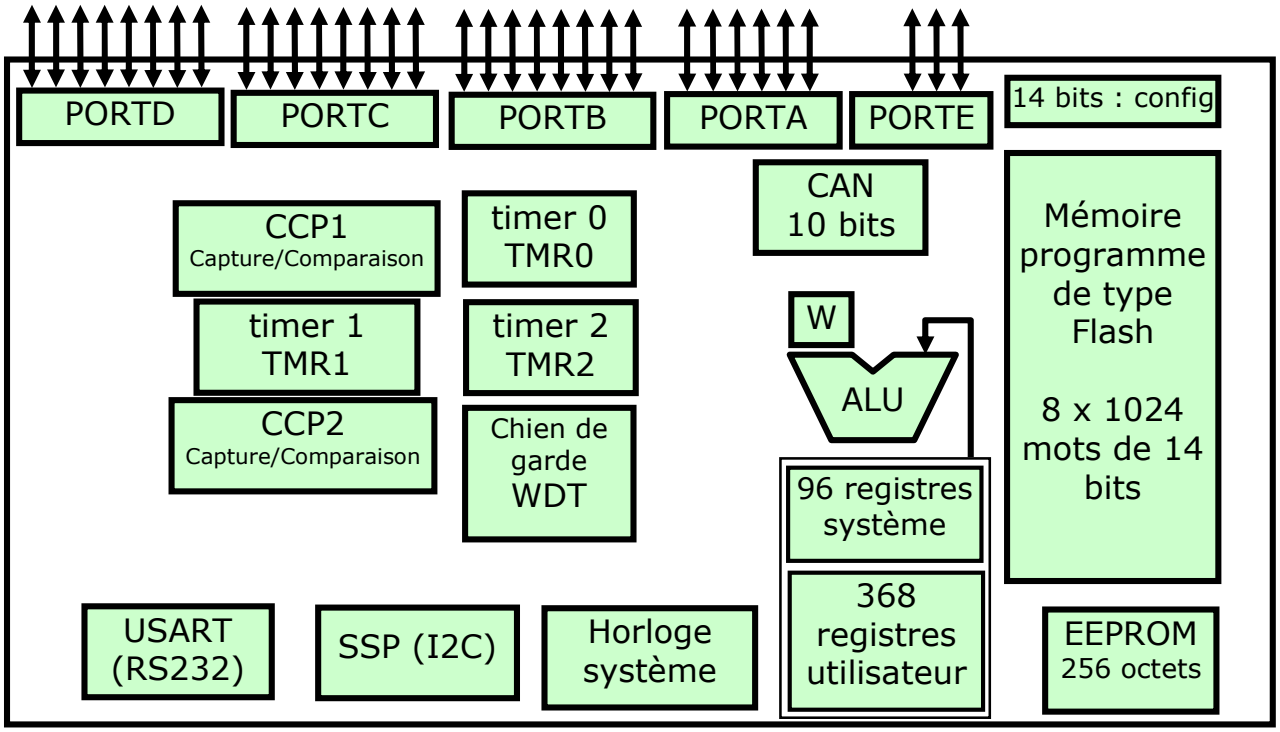


Fig. I.1 : Les éléments constitutifs du PIC 16F877

Le port D (8 bits) et le port E (3 bits) ne sont pas disponibles sur tous les processeurs. (Voir Tableau I.1)

II Les éléments de base du PIC 16F876/877

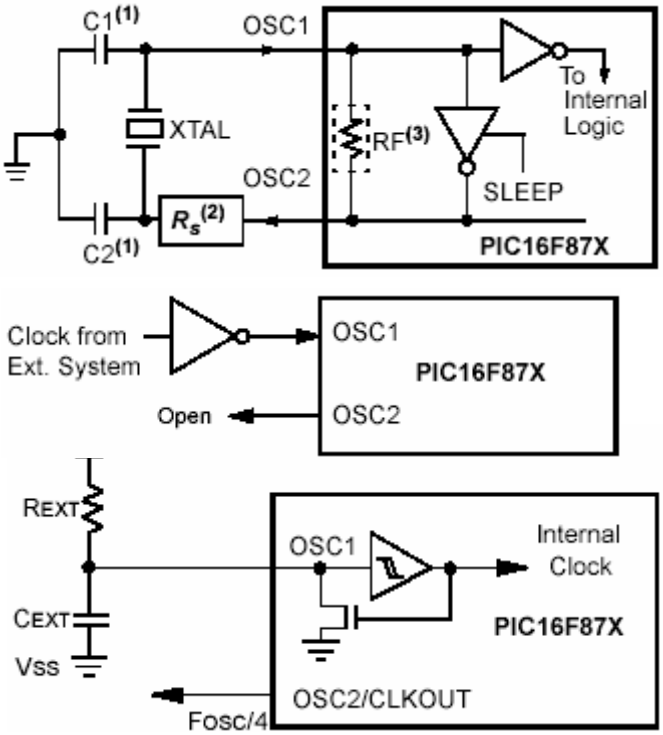
II.1 L'Horloge

L'horloge peut être soit interne soit externe. L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC.

Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 20 MHz selon le type de µC. Le filtre passe bas (Rs, C1, C2) limite les harmoniques dus à l'écrêtage et Réduit l'amplitude de l'oscillation, il n'est pas obligatoire.

Avec un oscillateur RC, la fréquence de l'oscillation est fixée par Vdd, Rext et Cext. Elle peut varier légèrement d'un circuit à l'autre.

Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser le PIC sur un processus particulier.



Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Dans la suite de ce document on utilisera le terme Fosc/4 pour désigner l'horloge système.

Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs.

II.2 L'ALU et l'accumulateur W

L'ALU est une Unité Arithmétique et logique 8 Bits qui réalise les opérations arithmétiques et logique de base. L'accumulateur W est un registre de travail 8 bits, toutes les opérations à deux opérandes passe par lui. On peut avoir :

- Une instruction sur un seul opérande qui est en général un registre situé dans la RAM
- Une instruction sur 2 opérandes. Dans ce cas, l'un des deux opérandes est toujours l'accumulateur W, l'autre peut être soit un registre soit une constante.

Pour les instructions dont un des opérandes est un registre, le résultat peut être récupéré soit dans l'accumulateur, soit dans le registre lui-même.

II.3 Organisation de la mémoire RAM

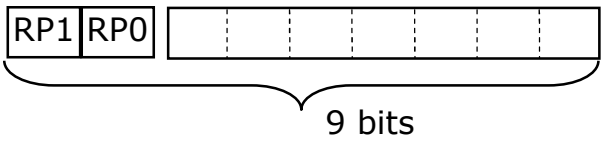
L'espace mémoire RAM adressable est de **512** positions de 1 octet chacune :

- 96 positions sont réservées au SFR (Special Function Registers) qui sont les registres de configuration du PIC.
- Les 416 positions restantes constituent les registres GPR (General Propose Registers) ou RAM utilisateur. Sur le 16F876 et 16F877, 3 blocs de 16 octets chacun ne sont pas implantés physiquement d'où une capacité de RAM utilisateur de 368 GPR.

Pour accéder à la RAM, on dispose de deux modes d'adressage :

II.3.1 Accès à la RAM par adressage DIRECT

Avec ce mode d'adressage, on précise dans l'instruction la valeur de l'adresse à laquelle on veut accéder. Par exemple, pour copier le contenu de l'accumulateur W dans la case mémoire d'adresse 50, on utilise l'instruction ***MOVWF 50***. Cette instruction sera codée sur 14 bits, la partie adresse est codée sur **7 bits** ce qui va poser quelques petits problèmes. En effet, 7 bits permettent d'adresser seulement 128 positions. Pour pouvoir adresser les 512 positions accessibles, il faut **9 bits** d'adresse. Pour avoir ces 9 bits, le PIC complète les 7 bits venant de l'instruction par deux bits situés dans le registre de configuration STATUS. Ces bits sont appelés RP0 et RP1 et doivent être positionnés correctement avant toute instruction qui accède à la RAM par l'adressage direct.



La RAM apparaît alors organisée en 4 banks de 128 octets chacun. L'adresse instruction permet d'adresser à l'intérieur d'un bank alors que les bits RP0 et RP1 du registre STATUS permettent de choisir un bank. *La* Figure II.1 *montre l'organisation de la RAM avec les zones allouée au SFR et aux GPR. Les zones hachurées ne sont pas implantées physiquement. Si on essaye d'y accéder, on est aiguillé automatiquement vers la zone [70_h,$7F_h$] appelée zone commune.*

Même si on précise une adresse supérieure à 127 (+ de 7 bits) dans une instruction, elle est tronquée à 7 bits puis complétée par les bits RP0 et RP1 pour former une adresse 9 bis. Par exemple, pour copier l'accumulateur W dans la case mémoire d'adresse $1EF_h$, il faut d'abord placer les bits RP0 et RP1 à 1 (bank 3), ensuite on utilise soit l'instruction **MOVWF 6Fh** soit l'instruction **MOVWF 1EFh**, qui donne le même résultat. En effet, que l'on écrive 6Fh = 0110 1111 ou 1EFh = 0001 1110 1111, le PIC ne prend que 7 bits soit : 1101111 = 6Fh et complète avec les bits RP1,RP0 pour obtenir 11 1101111 = 1EFh

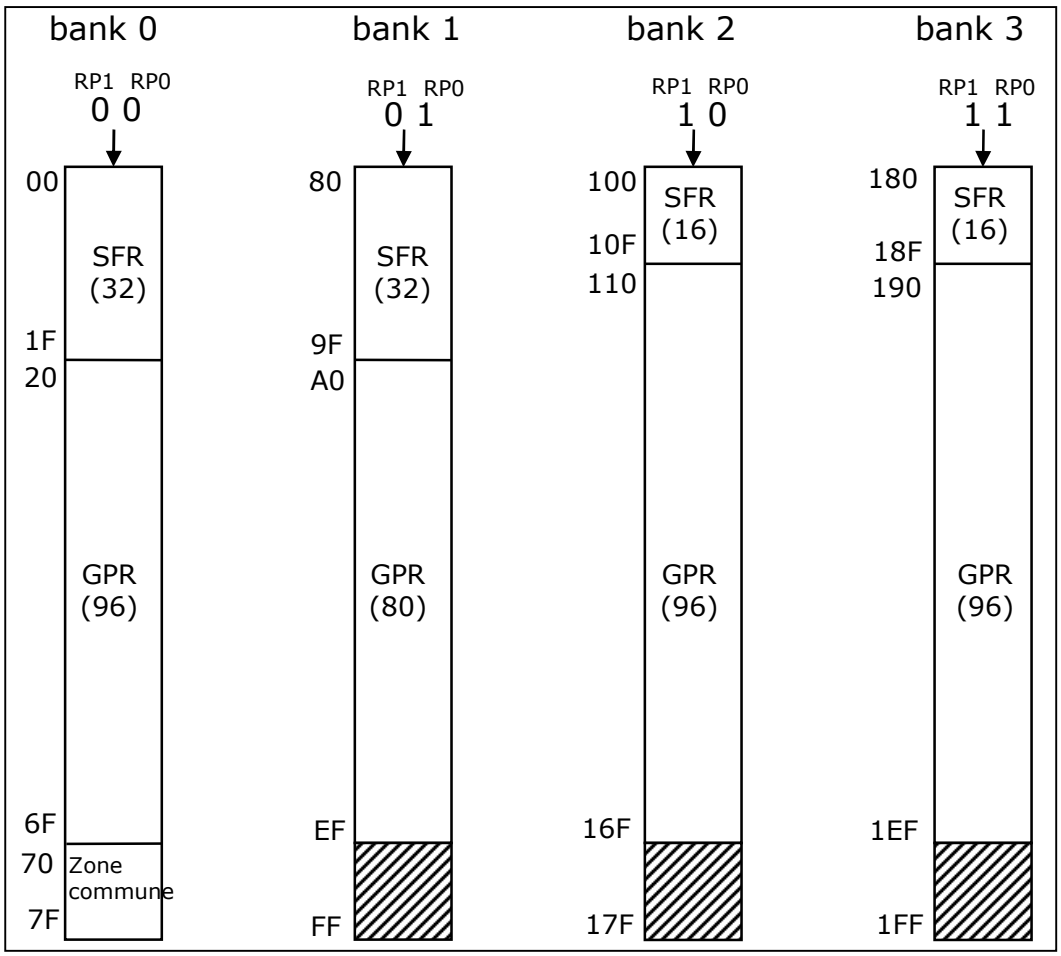


Figure II.1 : organisation de la RAM du 16F876/877

Nous allons anticiper un peu et présenter les instructions bcf et bsf et qui permettent de positionner un bit à 0 ou à 1

```
bcf     STATUS,RP0      ; place le bit RP0 à 0
bsf     STATUS,RP1      ; place le bit RP1  à 1
```

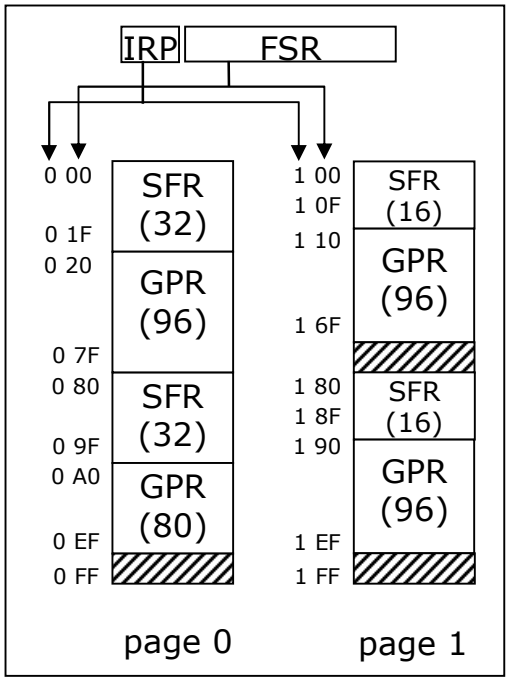
II.3.2 Accès à la RAM par l'adressage INDIRECT

Pour accéder à une position de la RAM en utilisant l'adressage indirect, on passe toujours par une position fictive appelée INDF. Exemple : l'instruction CLRF INDF signifie : mettre à zéro la case mémoire d'adresse INDF. Mais quelle est l'adresse de cette position appelée INDF ?

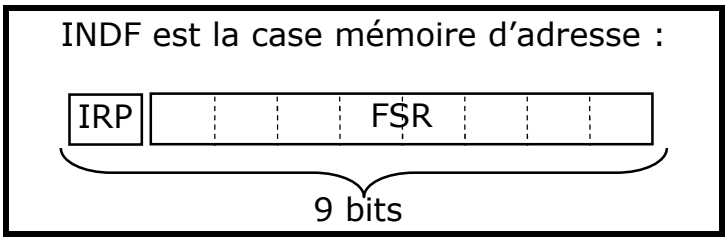
INDF est la case mémoire pointée par le registre (pointeur) FSR

Cela signifie que si on place 74h dans le registre FSR et ensuite on exécute l'instruction CLRF INDF, cela va remettre à zéro la case mémoire d'adresse 74h.

Ceci n'est pas tout à fait complet car (comme pour l'adressage direct) on va avoir un problème de capacité d'adressage. En effet, comme tous les registres, le registre de pointage FSR est un registre 8 bits, il peut donc adresser au maximum 256 positions mémoire (de 00h à FFh), c'est seulement la moitié de la RAM don on dispose. Il nous manque un bit pour avoir les 9 bits nécessaires. On utilise le bit IRP qui se trouve dans le registre STATUS.



Donc en résumé, chaque fois que le PIC rencontre le mot INDF dans un programme, il sait qu'il s'agit de la case mémoire dont l'adresse (9 bits) se trouve dans le registre FSR complété par le bit IRP du registre STATUS



IRP = 0 → 000h à 0FFh → page 0

IRP = 1 → 100h à 1FFh → page 1

II.4 Registres de configuration et leurs position dans la RAM

Bank 0		Bank 1		Bank 2		Bank 3	
00h	INDF	80h	INDF	100h	INDF	180h	INDF
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORD (*)	88h	TRISD (*)	108h		188h	
09h	PORT (*)	89h	TRISE (*)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	
10h	T1CON	90h					
11h	TMR2	91h	SSPCON2				
12h	T2CON	92h	PR2				
13h	SSPBUF	93h	SSPADD				
14h	SSPCON	94h	SSPSTAT				
15h	CCPR1L	95h					
16h	CCPR1H	96h					
17h	CCP1CON	97h					
18h	RCSTA	98h	TXSTA				
19h	TXREG	99h	SPBRG				
1Ah	RCREG	9Ah					
1Bh	CCPR2L	9Bh					
1Ch	CCPR2H	9Ch					
1Dh	CCP2CON	9Dh					
1Eh	ADRESH	9Eh	ADRESL				
1Fh	ADCON0	9Fh	ADCON1				

* PORTD et PORTE existent seulement dans le 16F877 et le 16F874

Tableau II.1 : Registres de configuration avec leurs adresses

II.5 Les registres de configuration avec leurs états après un RESET

STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
PIE2	N.I.	Réservé	N.I.	EEIE	BCLIE	N.I.	N.I.	CCP2IE	-r-0 0--0
PIR2	N.I.	Réservé	N.I.	EEIF	BCLIF	N.I.	N.I.	CCP2IF	-r-0 0--0
EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
CCPxCON	—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0	--00 0000
T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000
SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000
SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000
CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0
ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000
TRISx									1111 1111

Tableau II.2 : détail des registres SFR et leurs états au démarrage

II.6 Les instructions du 16F876/877

- Tous les PICs Mid-Range ont un jeu de 35 instructions,
- Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (OC) ainsi que l'opérande,
- Toutes les instructions sont exécutées en un cycle d'horloge, à part les instructions de saut qui sont exécutées en 2 cycles d'horloge. Sachant que l'horloge système est égale à fosc/4, si on utilise un quartz de 4MHz, on obtient une horloge fosc/4 = 1000000 cycles/seconde, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d' Instructions Par Seconde). Avec un quartz de 20MHz, on obtient une vitesse de traitement de 5 MIPS.

II.6.1 Les instructions « orientées Registre»

Ce sont des instructions qui manipulent un octet se trouvant dans la RAM. Ça peut être un registre de configuration SFR ou une case mémoire quelconque (Registre GPR)

II.6.2 Les instructions « orientées bits »

Ce sont des instructions destinées à manipuler directement un bit d'un registre que se soit un registre de configuration SFR ou une case mémoire quelconque (registre GPR). Tous les bits de la RAM peuvent être manipulé individuellement.

II.6.3 Les instructions opérant sur une constante

Ce sont les instructions entre l'accumulateur W est une constante K

II.6.4 Les instructions de saut et appel de procédures

Ce sont les instructions qui permettent de sauter à une autre position dans le programme et de continuer l'exécution du programme à partir de cette position.

II.6.5 Les indicateur d'état

Les bits Z, DC et C situés dans le registre STATUS sont des indicateurs qui permettent de savoir comment une instruction s'est terminée. Toutes les instructions n'agissent pas sur les indicateurs, voir liste des instructions ci-dessous.

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 4ème bits génère une retenue

Ces bits peuvent être utilisé très astucieusement par les instructions ***btfsc*** et ***btfss*** qui permettent de tester un bit et de réaliser un saut conditionnel. Nous aurons l'occasion d'en reparler dans la suite du cours.

STATUS	IRP	RP1	RP0			Z	DC	C

II.6.6 Le jeu d'instructions

{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**
ADDWF	**F,d**	W+F → {W,F ? d}	C,DC,Z	1
ANDWF	**F,d**	W and F → {W,F ? d}	Z	1
CLRF	**F**	Clear F	Z	1
COMF	**F,d**	Complémente F → {W,F ? d}	Z	1
DECF	**F,d**	décrémente F → {W,F ? d}	Z	1
DECFSZ	**F,d**	décrémente F → {W,F ? d} skip if 0		1(2)
INCF	**F,d**	incrémente F → {W,F ? d}	Z	1
INCFSZ	**F,d**	incrémente F → {W,F ? d} skip if 0		1(2)
IORWF	**F,d**	W or F → {W,F ? d}	Z	1
MOVF	**F,d**	F → {W,F ? d}	Z	1
MOVWF	**F**	W → F		1
RLF	**F,d**	rotation à gauche de F a travers C → {W,F ? d}	C	1
RRF	**F,d**	rotation à droite de F a travers C → {W,F ? d}		1
SUBWF	**F,d**	F – W → {W,F ? d}	C,DC,Z	1
SWAPF	**F,d**	permute les 2 quartets de F → {W,F ? d}		1
XORWF	**F,d**	W xor F → {W,F ? d}	Z	1

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

INSTRUCTIONS OPERANT SUR CONSTANTE				
ADDLW	**K**	W + K → W	C,DC,Z	1
ANDLW	**K**	W and K → W	Z	1
IORLW	**K**	W or K → W	Z	1
MOVLW	**K**	K → W		1
SUBLW	**K**	K – W → W	C,DC,Z	1
XORLW	**K**	W xor K → W	Z	1

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

On remarque que :

- les instructions qui agissent sur un registre ou un bit d'un registre contiennent toutes la lettre F dans le nom de l'instruction. Ceci vient du fait que chez Microchip, la RAM est appelée *register **F**ile* (**F**ichier des registres).
- Les instructions qui agissent sur une constante contiennent toutes la lettre L, parce que chez Microchip, on appelle « ***L**iteral* » ce genre d'adressage, chez d'autre constructeurs, on parle d'adressage immédiat

II.6.7 Les paramètres des instructions agissant sur registre

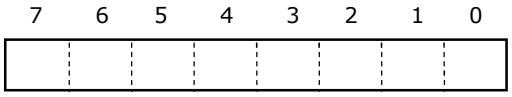
Pour les instructions qui agissent sur registre, le paramètre F représente l'adresse du registre considéré. Le paramètre d (destination) joue un rôle important, si on prend *d = 0 ou w*, le résultat de l'opération sera placé dans l'accumulateur W, si on prend *d = 1* ou *f*, le résultat de l'opération sera placé dans le registre précisé par F.

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

II.6.8 Les paramètres des instructions agissant sur bit

Pour les instructions agissant sur un bit, le paramètre F indique le registre qui contient le bit à modifier et le paramètre b indique le numéro du bit à modifier; on compte à partir de zéro en commençant à droite



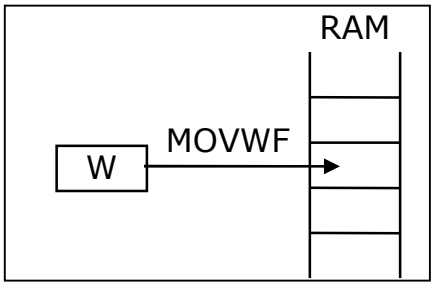
BSF STATUS,2 ; signifie : placer à 1 le bit 2 (3ème bit à partir de la droite) du registre STATUS

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

II.6.9 Les instruction MOVWF et MOVF

Ce sont les instructions les plus utilisées,

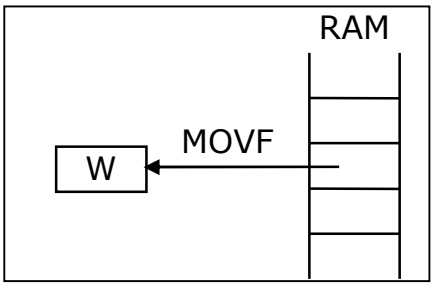
MOVWF permet de copier l'accumulateur W dans un registre (SFR ou GPR):



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

MOVF permet de copier le contenu d'un registre (SFR ou GPR) dans l'accumulateur W, le paramètre d doit être = 0




```
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
```

En réalité cette instruction peut s'avérer utile car, comme elle positionne l'indicateur Z, elle permet de tester si le contenu d'un registre est égal à zéro

II.6.10 Les instructions btfss et btfsc

Ces instructions permettent de tester un bit et de sauter ou non une ligne de programme en fonction de la valeur du bit,

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
…
```

II.6.11 Les instructions incfsz et decfsz

Ces instructions permette d'incrémenter ou de décrémenter un registre et de sauter si le résultat est nul

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'incrémentation 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.

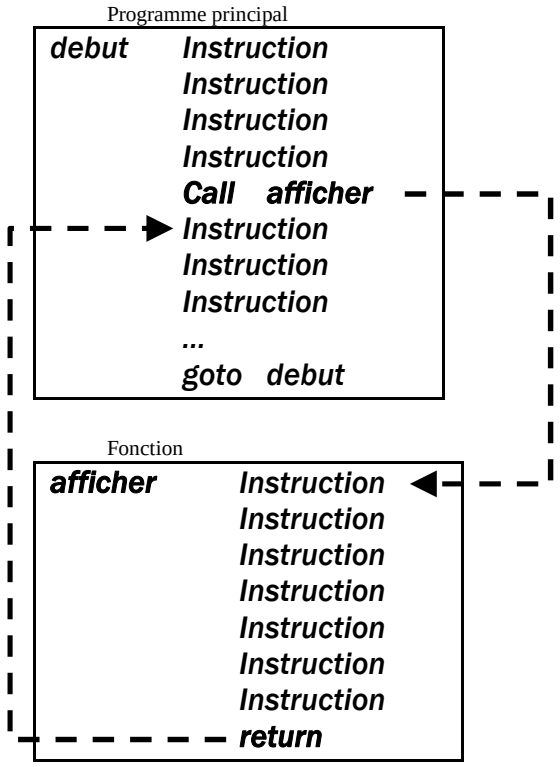
II.6.12 L'instruction goto

Permet de transférer l'exécution à une autre position du programme repérée par une ***étiquette*** (label)

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

II.6.13 L'instruction call

L'instruction ***call*** permet d'appeler une fonction. Une fonction est un sous programme écrit à la suite du programme principal. Sa première ligne doit comporter une étiquette et elle doit se terminer par ***return***



La différence en ***call*** et ***goto*** est que, quant le processeur rencontre l'instruction ***call***, il sauvegarde l'adresse de la ligne suivante avant d'aller exécuter les instructions constituant la fonction. Comme ça, quand il rencontre l'instruction ***return***, il sait où il doit retourner pour continuer l'exécution du programme principal.

II.7 Les directives de l'assembleur MPASM

Les directives de l'assembleur sont des instructions qu'on ajoute dans le programme et qui seront interprétées par l'assembleur MPASM. Ce ne sont pas des instructions destinées au PIC. Nous ne présentons ici que quelques directives, pour le reste, consulter la documentation de MPASM *"MPASM USER'S GUIDE"*

- **LIST** : permet de définir un certain nombre de paramètres comme le processeur utilisé (p), la base par défaut pour les nombres (r) ainsi que d'autres paramètres. Exemple :
  LIST p=16F876, r=dec

avec r=dec, les nombres sans spécification particulière seront considérés par l'assembleur comme des nombre décimaux, sinon voir tableau ci dessous

Base	Préfixe	Exemple (36)
Décimal	D'nnn' .nnn	D'36' .36
Hexadécimal	H'nn' 0xnn nnh	H'24' 0x24 24h
Binaire	B'….'	B'00100100'
Octal	O'nnn'	O'44'

- **INCLUDE** : permet d'insérer un fichier source. Par exemple le fichier p16f876.inc contient la définition d'un certain nombre de constante comme les noms des registres ainsi que les noms de certain bits;
  INCLUDE "p16f876.inc"

- **EQU** : permet de définir une constante ou une variable :
  XX EQU 0x20
  Chaque fois que le compilateur rencontrera XX, il la remplacera par 0x20. Ça peut être une constante s'il s'agit d'une instruction avec adressage Literal, ou d'une adresse s'il s'agit d'une instruction avec adressage direct.

- **ORG** : définit la position dans la mémoire programme à partir de laquelle seront inscrites les instructions suivantes.

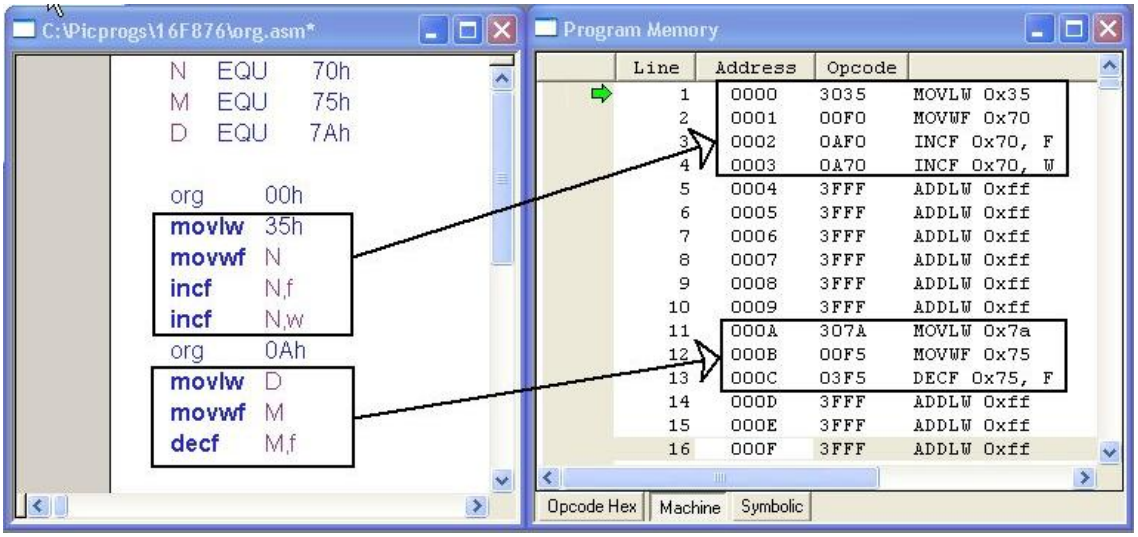


Fig. II.1 : Illustration de la directive ORG

- DE : pour déclarer des donnés qui seront stockée dans l'EEPROM de donnée au moment de l'implantation du programme sur le PIC
  ORG 0x2100
  DE "Programmer un PIC, rien de plus simple", 70, 'Z'

- **DT** : pour déclarer un tableau RETLW
  DT 10,35h,'Hello' ; sera remplacée par la suite d'instructions :

  RTLW 10
  RTLW 35h
  RTLW 'H'
  RTLW 'e'
  RTLW 'l'
  RTLW 'l'
  RTLW 'o'

- **END** : indique la fin du programme

- **__CONFIG** : permet de définir les 14 fusibles de configuration qui seront copiés dans l'EEPROM de configuration (adresse 2007h) lors de l'implantation du programme dans le PIC,

CP1	CP0	DEBUG	—	WRT	CPD	LVP	BODEN	CP1	CP0	PWRTE	WDTE	F0SC1	F0SC0

CP1/CP0 : bits 13/12 ; Déterminent quelle zone de la mémoire programme sera protégée contre la lecture externe (via ICSP) ou l'écriture par programme conformément à l'état du bit 9 (WRT). On peut choisir de protéger la totalité de la mémoire ou seulement une partie. Les différentes zones pouvant être protégées sont les suivantes :
1 1 : Aucune protection (_CP_OFF)
1 0 : Protection de la zone 0x1F00 à 0x1FFF (_CP_UPPER_256)
0 1 : Protection de la zone 0x1000 à 0x1FFF (_CP_HALF)
0 0 : Protection de l'intégralité de la mémoire (_CP_ALL)

DEBUG : bit 11 : Debuggage sur circuit. Permet de dédicacer RB7 et RB6 à la communication avec un debugger.
1 : RB6 et RB7 sont des I/O ordinaires (_DEBUG_OFF)
0 : RB6 et RB7 sont utilisés pour le debuggage sur circuit (_DEBUG_ON)

WRT : bit 9 : Autorisation d'écriture en flash
1 : Le programme peut écrire dans les zones non protégées par les bits CP1/CP0 (_WRT_ENABLE_ON)
0 : Le programme ne peut pas écrire en mémoire flash (_WRT_ENABLE_OFF)

CPD : bit 8 : Protection en lecture de la mémoire EEPROM de données.
1 : mémoire EEPROM non protégée (_CPD_OFF)
0 : mémoire EEPROM protégée contre la lecture externe via ICSP (_CPD_ON)

LVP : bit 7 : Utilisation de la pin RB3/PGM comme broche de programmation 5V
1 : La pin RB3 permet la programmation du circuit sous tension de 5V (_LVP_ON)
0 : La pin RB3 est utilisée comme I/O standard (_LVP_OFF)

BODEN : bit 6 : provoque le reset du PIC en cas de chute de tension (surveillance de la tension d'alimentation)
1 : En service (_BODEN_ON)
0 : hors service (_BODEN_OFF)

PWRTE : bit 3 : Délai de démarrage à la mise en service. Attention, est automatiquement mis en service si le bit BODEN est positionné.
1 : délai hors service (sauf si BODEN = 1) (_PWRTE_OFF)
0 : délai en service (_PWRTE_ON)

WDTE : bit 2 : Validation du Watchdog timer
1 : WDT en service (_WDT_ON)
0 : WDT hors service (_WDT_OFF)

FOSC1/FOSC0 : bits 1/0 : sélection du type d'oscillateur
11 : Oscillateur de type RC (_RC_OSC) (3K < R < 100k, C > 20 pF)
10 : Oscillateur haute vitesse (_HS_OSC) (4 Mhz à 20 Mhz)
01 : Oscillateur basse vitesse (_XT_OSC) (200 kHz à 4 Mhz)
00 : Oscillateur faible consommation (_LP_OSC) (32 k à 200 kHz)

Voici 3 exemples d'utilisation :

__CONFIG B'11111100111001'

__CONFIG H'3F39'

__CONFIG _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_ON & _CPD_OFF & _LVP_OFF & _BODEN_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC

II.8 Illustrant de quelques aspects de programmation :

Exercice 1) : analyse de programmes

Analyser (tracer) le programme ci-dessous :

```
clrf    35h
Incf    35h,f
Incf    35h,w

clrf    135h
Incf    135h,f
Incf    135h,w

movlw   100
movwf   70h
movlw   50
movwf   71h
movf    70h,w
addwf   71h,f
```

Exercice 2) : Accès à la RAM par l'adressage direct

Donner le programme qui copie :

35 dans la position 20h, 'A' dans la position A0h
-5 dans la position 110h, 35h dans la position 190h

Exercice 3) : Accès à la RAM par l'adressage indirect

Donner le programme qui copie l'alphabet majuscule dans la RAM à partir de la position 190h

Exercice 4) : Comparaison et branchement conditionnel

Comparer les contenus des cases mémoire 6Fh et EFh, s'il son égaux mettre à zéro tous les bits de la case 16Fh sinon mettre à 1 tous les bits de la case 1EFh

Exercice 5) : Soustraction

Donner le programme qui :

- soustrait la constante 33 de l'accumulateur W (W-33)
- Soustrait la constante 40h de la case mémoire d'adresse 70h ([70h]-40h)
- qui soustrait le contenu de la case mémoire 70h de l'accumulateur W avec le résultat dans W
  (W – [70h] → W)
- qui soustrait le contenu de la case mémoire 71h de l'accumulateur W avec le résultat dans la case mémoire
  (W – [71h] → [71h])

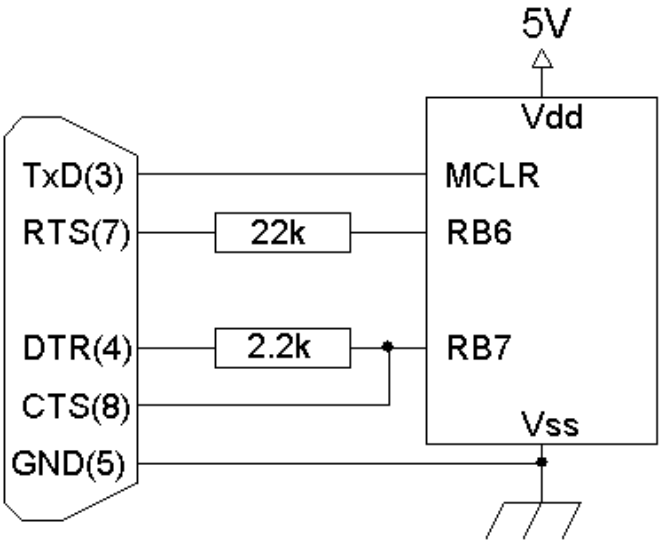
III Les outils de développement

Les étapes nécessaires permettant de voir un programme s'exécuter sur un PIC sont :

- Ecrire un programme en langage assembleur dans un fichier texte et le sauvegarder avec l'extension **.asm**
- Compiler ce programme avec l'assembleur MPASM fourni par Microchip. Le résultat est un fichier exécutable avec l'extension **.hex** contenant une suite d'instruction compréhensible par le pic.
- Transplanter le fichier **.hex** dans la mémoire programme du PIC (mémoire flash) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de Microchip ou tout autre programmateur acheté ou réalisé par soit même.
- Mettre le PIC dans son montage final, mettre sous tension et admirer le travail.

Microchip propose gratuitement l'outil de développement **MPLAB** qui regroupe l'éditeur de texte, le compilateur MPASM, un outil de simulation et le logiciel de programmation. Le programmateur lui-même, n'est malheureusement pas gratuit.

Pour ce qui nous concerne, nous utiliseront MPLAB pour écrire, compiler et éventuellement simuler nos programmes, ensuite, pour implanter les programmes dans la mémoire flash du PIC, nous utiliserons un programmateur fait maison que nous piloteront par le logiciel ICPROG, les deux sont disponibles gratuitement sur le Web.



Programmeur à 2 résistances

Internet est riche en informations sur ce sujet. Vous trouverez une rubrique sur les programmateurs de PIC sur mon site http://z.oumnad.123.fr

III.1 Deux mots sur MPLAB

MPLAB-IDE peut être téléchargé sur le site Web http://www.microchip.com

Après l'installation, lancer MPLAB et faire les config ci-dessous :

- Configure → Select Device → PIC16F876 ou PIC16F877

Nous allons réaliser un tout petit programme sans grand intérêt pour voir la procédure de fonctionnement (avec MPLAB 6.30)

- Ouvrir une nouvelle fenêtre (de l'éditeur) pour commencer à écrire un programme : **file → new** ou cliquez sur l'icône feuille blanche
- Taper le petit programme ci-dessous dans la fenêtre qui vient de s'ouvrir. Ce programme incrémente sans fin la position mémoire (RAM) 70_H

```
loop    incf 70h,1
        goto looop
        end
```

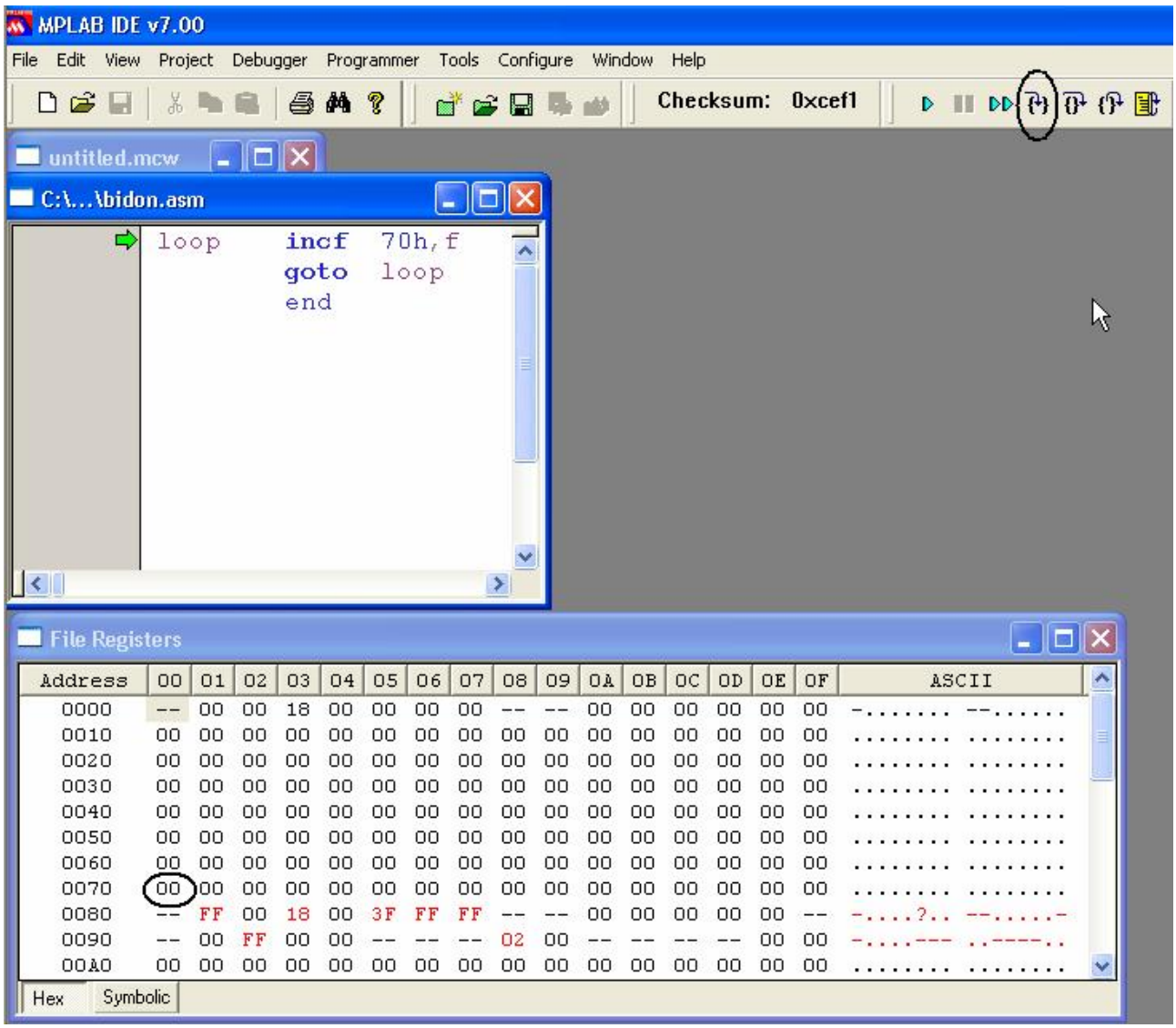
- Sauvegarder (file → save) ce programme dans la directory de votre chois sous le nom *bidon.asm*
- Lancer la compilation du programme à l'aide de la commande ***project → Quikbuild***
  Apparemment il y a un problème, le compilateur nous dit qu'il y une erreur à la ligne 2

  Error[113] C:\...\BIDON.ASM 2 : Symbol not previously defined (looop)

  *E*videmment, le label loop définit dans la ligne précédente prend seulement deux o. Corrigez et recommencez. Cette fois ça a l'air d'aller. On peut vérifier que le compilateur a crée le fichier *bidon.hex* dans la même directory où se trouve *bidon.asm*. Les fichiers *bidon.cod, bidon.err et bidon.lst* ne nous servent à rien pour l'instant on peut les détruire.

- Nous pouvons maintenant exécuter notre programme en simulation pour voir s'il réalise bien la tache demandée :
  Pour faire apparaître la barre d'outil du simulateur :
  Debugger → Select tool → MPLAB SIM
- Ouvrez la fenêtre qui visualise la mémoire RAM : view → *FileRegisters*. et repérer la case mémoire 70h
- Exécuter maintenant le programme PAS à PAS en cliquant à chaque fois sur le bouton Step Into **{↴}** en observant la case mémoire 70h . (on dirait que ça marche).
- On peut aussi exécuter en continu en cliquant sur le bouton animate ⏩ , pour arrêter, il faut cliquer sur le bouton halt ⏸

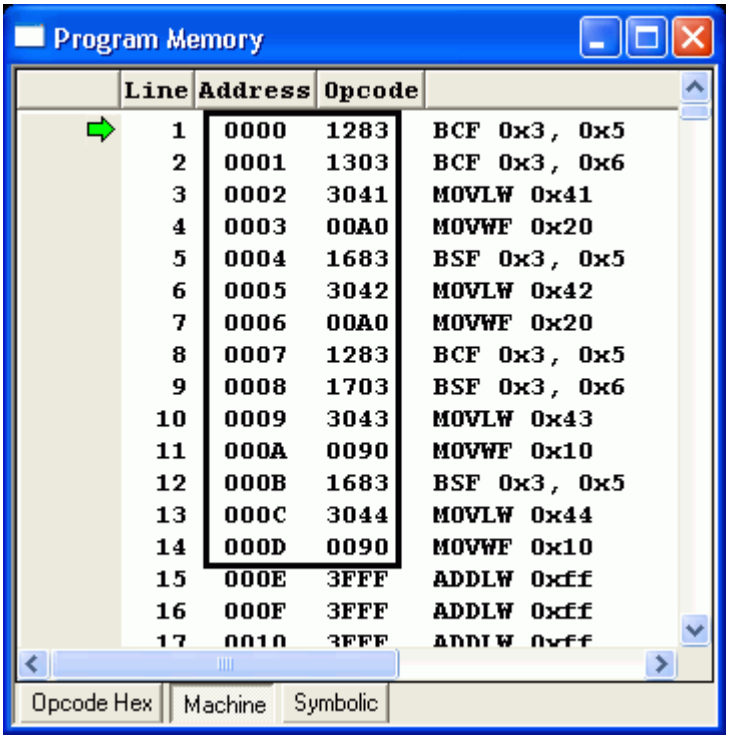
Pour plus de détail, consulter le manuel d'utilisation de MPLAB


MPLAB IDE v7.00
File Edit View Project Debugger Programmer Tools Configure Window Help
Checksum: 0xcef1
untitled.mcw
C:\...\bidon.asm
loop incf 70h,f
goto loop
end
File Registers
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
0000 -- 00 00 18 00 00 00 00 -- -- 00 00 00 00 00 00 -....... --.....
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0080 -- FF 00 18 00 3F FF FF -- -- 00 00 00 00 00 -- -....?.. --.....-
0090 -- 00 FF 00 00 -- -- -- 02 00 -- -- -- -- 00 00 -....--- ..----..
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
Hex
Symbolic

III.2 Exemple de programme

```
;*****************************************************************
; Ce programme écrit les lettres A, B, C, D dans les positions suivantes de la RAM
; A  =>  20h                B => A0h              C => 110h          D => 190h
;*****************************************************************

        list p=16f877 , r=dec
        include <p16f877.inc>
        __CONFIG  _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
                  _LVP_OFF & _BODEN_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC

        bcf             STATUS,RP0      ;bank 0
        bcf             STATUS,RP1      ;bank 0
        movlw           'A'
        movwf           20h

        bsf             STATUS,RP0      ;bank 1
        movlw           'B'
        movwf           0xA0

        bcf             STATUS,RP0      ;bank 2
        bsf             STATUS,RP1      ;bank 2
        movlw           'C'
        movwf           110h

        bsf             STATUS,RP0      ;bank 3
        movlw           'D'
        movwf           190h

        end
```

Compilons le programme avec MPLAB *(project→quikbuild)* et visualisons la mémoire programme *(view→program memory)* , on obtient la fenêtre ci-dessus.



Observations :

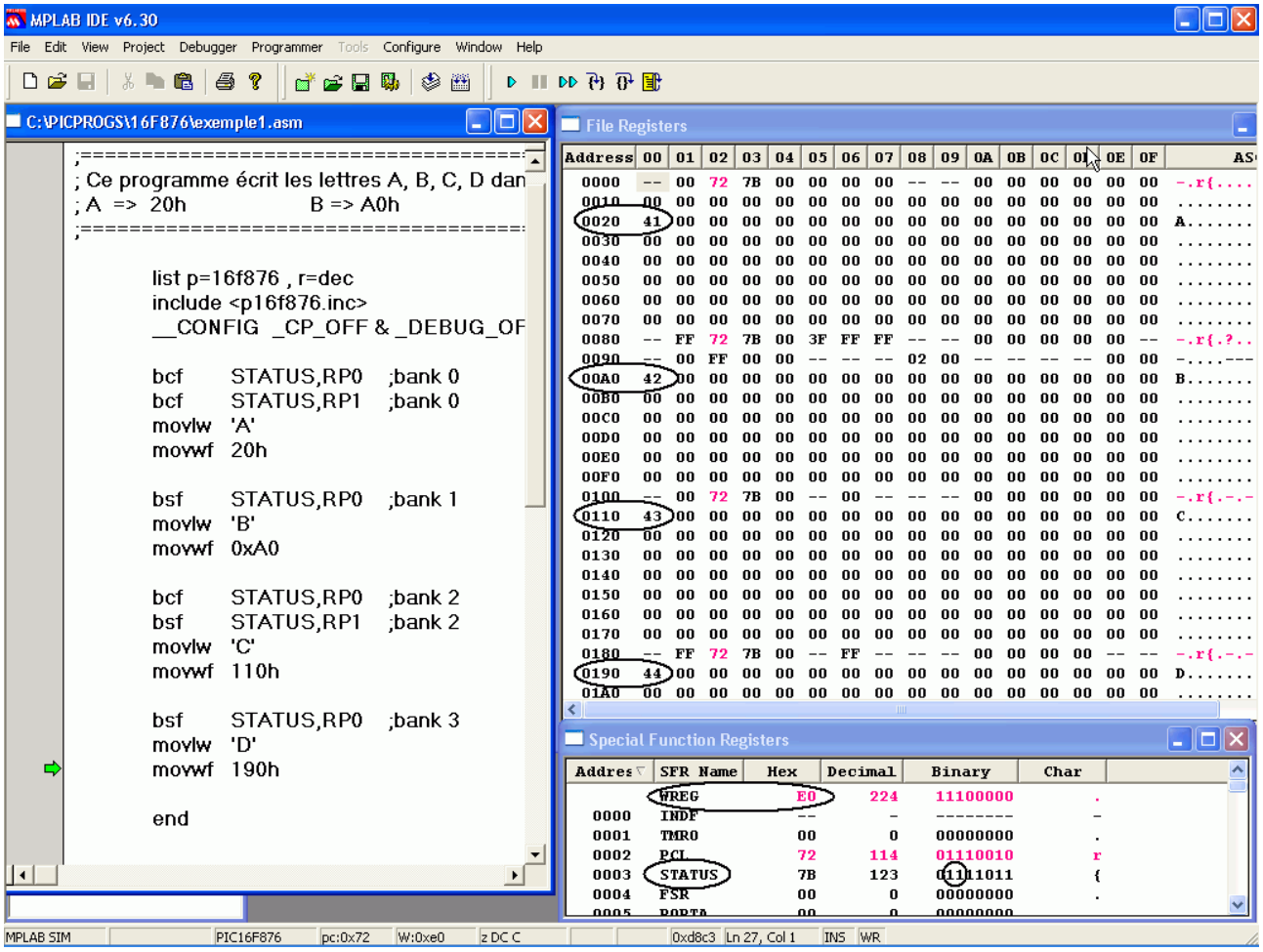
- Comme il n'y a pas d'interruption, on ne s'est pas soucié de l'implantation du programme dans la mémoire programme (directive ORG). on remarque sur la fenêtre que l'implantation du programme commence à l'adresse 0.

- La colonne de droite de cette fenêtre obtenue par cross-assemblage du code machine n'utilise pas les noms des registres mais leurs adresses. Si nous avons pu utiliser les instruction comme *bcf STATUS,RP0*, au lieu de *bcf 0x3,0x5* c'est parce que le fichier *p16f876.inc* que nous avons inclus contient les déclarations *STATUS EQU 0x3* et *RP0 EQU 0x5*

- On remarque aussi que l'instruction *movwf 190h* a été remplacée par *movwf 0x10*, c'est tout à fait normal car on ne retient que 7 bits de l'adresse 190h = 0001 1001 0000 ce qui donne 001 0000 = 10h. Rassurons nous, c'est quand même l'adresse 190h qui sera adressée l'adresse 001 0000 sera complétée par RP1,RP0=11 (bank 3) et on obtient 11001 0000 = 190h

Simulation :

Visualisons la RAM *(view→file registers)* et les registres de configuration *(vew→Special function registers)* et exécutons le programme pas par pas tout en observant l'accumulateur WREG, le registre STATUS et les cases mémoire concernées.



III.3 Faire des boucles de temporisation

Il arrive souvent qu'on désire introduire des temporisations pendant l'exécution d'un programme. Le PIC dispose de 3 *timers* permettant de gérer le temps avec précision. Nous étudierons ces modules plus tard, pour l'instant regardons comment on peut réaliser des temporisations à l'aide de simples boucles. L'idée est d'initialiser une variable à une valeur donnée et ensuite la décrémenter en boucle jusqu'à ce qu'elle atteint 0. Connaissant le temps d'exécution de chaque instruction, on peut calculer le temps que mettra le processeur terminer la boucle de décrémentation

III.3.1 Temporisation avec une boucle

Examinons l'exemple ci-dessous, on met une valeur N1 dans la case mémoire 70h et on la décrémente jusqu'à 0

```
        movlw   4
        movwf   70h
ici     decfsz  70h,f
        goto    ici
```

- Les instructions *movlw* et *movwf* prennent 1 cycle chacune
- L'instruction *decfsz* prend un cycle si elle ne saute pas et 2 cycles quand elle saute
- L'instruction *goto* prend 2 cycles
- chaque passage dans la boucle prend (1+2) cycle sauf le dernier qui prend 2 cycle

$$T = 2 + (N1-1)*3 + 2 = 3*N1 + 1 \text{ cycles}$$

La valeur max que l'on peut donner à N1 est = 0 = 256, ce qui donne une temporisation max de 769 cycles.

Avec un quartz = fosc = 4 Mhz, 1 cycle = fosc/4 = 1 µs, ce qui donne une temporisation max de 769 µs

Pour faciliter l'utilisation de cette temporisation on va l'utiliser comme une fonction que l'on appellera ***tempo1.*** On note AN1 la case mémoire qui sert de compteur, il faut la déclarer au début avec la directive ***EQU***

```
tempo1: movlw   valeur N1 ; cette ligne peut être placée dans le programme principal
        movwf   AN1
t1:     decfsz  AN1,f
        goto    t1
        return
```

Cette fonction doit être placée après le programme principal, et on l'appelle avec l'instruction ***call tempo1***

Pour le calcul il faut rajouter 2 cycles pour l'instruction ***call*** et 2 cycles pour l'instruction ***return***, ce qui donne

$$T1 = 3\ N1 + 5 \text{ cycles}$$

Le maximum est obtenu pour N1 =256, soit T1max = 773 µs = 0.77 ms

III.3.2 Temporisation avec 2 boucles imbriquées

La boucle intérieure (N1) se fait toujours 256 fois. La boucle extérieure se fait N2 fois

```
tempo2 :    movlw   valeur de N2  ; cette ligne peut être placée dans le programme principal
            movwf       AN2
t2 :        decfsz       AN1,f
            goto        t2
            decfsz      AN2,f
            goto        t2
            return
```

T2 = 2 + 2 + [N2(t1+3)-1] + 2 avec t1=256*3-1=767 on obtient :

$$T2 = 770\ N2 + 5$$

Le maximum est obtenu pour N2 =256, soit T2max = 197125 µs = 0.197 s

III.3.3 Temporisation avec 3 boucles imbriquées

Les boucles intérieures (N1 et N2) se font toujours 256 fois. La boucle extérieure se fait N3 fois

```
tempo3 :    movlw   valeur de N3 ; cette ligne peut être placée dans le programme principal
            movwf       AN3
t3 :        decfsz      AN1,f
            goto        t3
            decfsz      AN2,f
            goto        t3
            decfsz      AN3,f
            goto        t3
            return
```

T3 = 2 + 2 + N3(t2+3)-1 + 2 avec t2=256*(767+3)-1 on obtient :

$$T3 = 197122\ N3 + 5$$

Le maximum est obtenu pour N3 =256, soit T3max = 50463237 µs = 50.46 s

Remarque : La précision de ces fonctions peut être améliorée en y insérant des instructions ***nop***, dans ce cas il faut revoir les formules.

IV Les autres modules du 16F876/877

IV.1 Le port d' E/S PORTA

Le port A désigné par PORTA est un port de 6 bits (RA0 à RA5). RA6 et RA7 ne sont pas accessibles.

La configuration de direction se fait à l'aide du registre TRISA, positionner un bit de TRISA à 1 configure la broche correspondante de PORTA en entré et inversement. Au départ toutes les broches sont configurées en entrée

IV.1.1 La broche RA4

En entrée, la broche RA4 peut être utilisée soit comme E/S numérique normale, soit comme entrée horloge pour le Timer TMR0

En sortie, RA4 est une E/S à drain ouvert, pour l'utiliser comme sortie logique, il faut ajouter une résistance de pull-up externe. Le schéma (Fig. IV.1) illustre (pour les non électroniciens) le principe d'une sortie drain ouvert (ou collecteur ouvert) : si RA4 est positionnée à 0, l'interrupteur est fermé, la sortie est reliée à la masse, c'est un niveau bas. Si RA4 est placée à 1, l'interrupteur est ouvert, la sortie serait déconnectée s'il n'y avait pas la résistance externe qui place la sortie au niveau haut.

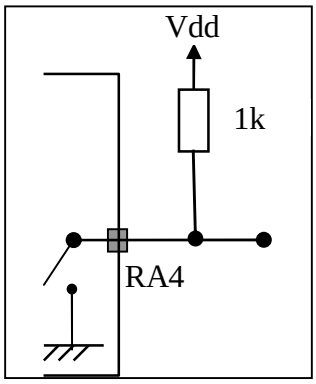


Fig. IV.1 : résistance de pull-up

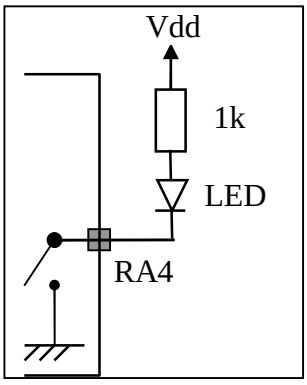


Fig. IV.2 : LED sur RA4

Si on veut utiliser RA4 pour allumer une LED, on peut utiliser le schéma de *Fig. IV.2*. Il faut juste remarquer que la logique est inversée, si on envoie 0 sur RA4, l'interrupteur se ferme et la LED s'allume. Si on envoie 1, l'interrupteur s'ouvre et la LED s'éteint.

Le schéma illustré sur *Fig. IV.3* peut aussi être utilisé. La logique n'est pas inversée mais il demande une précaution particulière. Il ne faut pas positionner la sortie RA4 à l'aide d'une instruction qui réalise une opération sur l'état actuel du port; genre IORWF, ANDWF, XORWF ou COMF, ADDWF, INCF … Ces instructions réalisent une lecture-écriture en commencent par lire l'état du port pour ensuite faire une opération dessus. Or, (sur *Fig. IV.3*) si la sortie était au niveau haut, l'interrupteur est ouvert, la LED est allumée et elle impose une tension de l'ordre de 1.5V qui sera considérée (à tort) comme un niveau bas lors de la lecture du port par les instructions précitées. La solution est d'utiliser des instructions qui positionnent le PORT sans tenir compte de son état courant comme MOVWF, BSF ou BCF

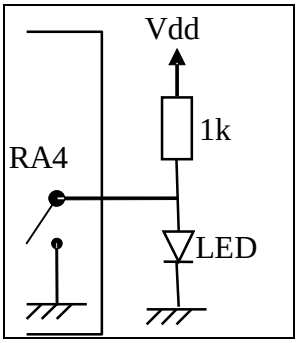


Fig. IV.3 : LED sur RA4

IV.1.2 Les autres broches

Les autres broches (RA0, RA1, RA2, RA3 et RA5) peuvent être utilisée soit comme E/S numériques soit comme entrées analogiques. Au RESET, ces E/S sont configurées en entrées **analogiques**. Pour les utiliser en E/S numériques, il faut écrire '00000110' dans le registre ADCON1 (pour plus de détail, voir chapitre IV.14).

Pour utiliser PORTA en port Numérique (normal), il faut placer 06h dans le registre ADCON1 (bank1)

Quelque soit le mode (Analogique ou Numérique), il faut utiliser le registre TRISA pour configurer la direction des E/S :

- Bit *i* de TRISA = 0 → bit *i* de PORTA configuré en **sortie**
- Bit *i* de TRISA = 1 → bit *i* de PORTA configuré en **entrée**

OPTION_REG :	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
ADCON1 :	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0

IV.2 Le port d' E/S PORTB

- Le port B désigné par PORTB est un port bidirectionnel de 8 bits (RB0 à RB7). Toutes les broches sont compatibles TTL.
- La configuration de direction se fait à l'aide du registre TRISB, positionner un bit de TRISB à 1 configure la broche correspondante de PORTB en entré et inversement. Au départ toutes les broches sont configurées en entrée.
- En entrée, la ligne RB0 appelée aussi INT peut déclencher l'interruption externe INT.
- En entrée, une quelconque des lignes RB4 à RB7 peut déclencher l'interruption RBI. Nous reviendrons là-dessus dans le paragraphe réservé aux interruptions.

IV.3 Le port d' E/S PORTC

- Le port C désigné par PORTC est un port bidirectionnel de 8 bits (RC0 à RC7). Toutes les broches sont compatibles TTL.
- La configuration de direction se fait à l'aide du registre TRISC, positionner un bit de TRISC à 1 configure la broche correspondante de PORTC en entré et inversement. Au départ toutes les broches sont configurées en entrée.
- Toutes les broches du port C peuvent être utilisées soit comme E/S normales soit comme broches d'accès à différents modules comme le timer 1, les modules de comparaison et de capture CCP1/2, le timer 2, le port I2C ou le port série, ceci sera précisé au moment de l'étude de chacun de ces périphériques.
- Pour l'utilisation d'une broche du port C comme E/S normale, il faut s'assurer qu'elle n'a pas été affectée à un de ces modules. Par exemple, si

TIMER1 est validé, il peut utiliser les broches RC0 et RC1 selon sa configuration.

IV.4 Le port d' E/S PORTD

- Le port D désigné par PORTD est un port bidirectionnel de 8 bits (RD0 à RD7). Toutes les broches sont compatibles TTL et ont la fonction trigger de Schmitt en entrée.
- Chaque broche et configurable en entrée ou en sortie à l'aide du registre TRISD. Pour configurer une broche en entrée, on positionne le bit correspondant dans TRISD à 1 et inversement.
- PORTD n'est pas implémenté sur le 16F876, il est disponible sur le 16F877
- PORTD peut être utilisé dans un mode particulier appelé *parallel slave port*, pour cela il faut placer le bit 4 (*PSPMODE*) de TRISE à 1. Dans ce cas les 3 bits de PORTE deviennent les entrées de control de ce port (RE, WE et CS)

Pour utiliser PORTD en mode normal, il faut placer le bit 4 de TRISE à 0

IV.5 Le port d' E/S PORTE

- PORTE contient seulement 3 bits RE0, RE1 et RE2. Les 3 sont configurables en entrée ou en sortie à l'aide des bits 0, 1 ou 2 du registre TRISE.
- PORTE n'est pas implémenté sur le 16F876, il est disponible sur le 16F877
- Les 3 bits de PORTE peuvent être utilisés soit comme E/S numérique soit comme entrées analogiques du CAN. La configuration se fait à l'aide du registre ADCON1.
- Si le bit 4 de TRISE est placé à 1, Les trois bits de PORTE deviennent les entrées de control du PORTD qui (dans ce cas) fonctionne en mode parallel Slave mode
- A la mise sous tension (RESET), les 3 broche de PORTE sot configuré en entrée analogique.

Pour utiliser les broches de PORTE en E/S numériques normales : - Placer 06h dans ADCON1 - Placer le bit 4 (*PSPMODE*) de TRISE à 0

Exercice 6) : Clignoter une LED

Donner le programme qui fait clignoter une LED branchée sur RA0 avec une temporisation voisine de 0.5s. Sachant que le PIC est doté d'un quartz de 4 MHz, la temporisation sera réalisée à l'aide de boucles imbriquées

Exercice 7) : compteur impulsions

Programme qui :

- place la sortie RB0 à 1
- compte 150 impulsions sur l'entrée RA4
- remettre la sortie RB0 à 0

IV.6 La mémoire EEPROM de données

Le PIC 16F876/877 dispose de 256 octets de mémoire EEPROM de donnée. Son implantation physique commence à la position d'adresse absolue 2100h. Mais pour y accéder à partir des programmes utilisateur on utilise l'adressage relatif par rapport à la première position. La première position aura l'adresse 0, la deuxième aura l'adresse 1. . . et la dernière aura l'adresse 255.

Pour accéder à la EEPROM, on utilise 4 registres particuliers :

- EEADR : registre d'adresse (relative) (bank 2)
- EEDATA : registre de donnée (bank 2)
- EECON1 : registre de control (bank 3)
- EECON2 : 2ème registre de control (bank 3)

Le registre EECON1 :

EEPGD	—	—	—	WRERR	WREN	WR	RD

EEPGD : Accès à la mémoire EEPROP ou à la mémoire Programme
0 : EEPROM de données
1 : Mémoire programme (flash)

WRERR : Erreur d'écriture (indicateur)
0 : Pas d'erreur
1 : Une erreur s'est produite

WREN : Validation de l'écriture dans l'EEPROM
0 : Ecriture interdite
1 : Ecriture autorisée

WR : Write Enable. Ce bit doit être mis à 1 pour démarrer l'écriture d'un octet. Il est remis à zéro automatiquement à la fin de l'écriture. Ce bit ne peut pas être mis à zéro par une instruction.

RD : Read Enable. Ce bit doit être mis à 1 pour démarrer la lecture d'un octet. Il est remis à zéro automatiquement à la fin de la lecture. Ce bit ne peut pas être mis à zéro par une instruction

IV.6.1 Procédure de lecture dans l'EEPROM

Pour lire le contenu d'une position de la mémoire EEPROM, on place l'adresse dans le registre EEADR, on lance l'opération de lecture à l'aide du bit RD du registre EECON1, la donnée désirée est tout de suite disponible dans le registre EEDATA :

1) Mettre le bit EEPGD à 0 pour pointer sur l'EEPROM de donnée
2) Placer l'adresse relative de la position à lire dans EEADR
3) Mettre le bit RD à 1 pour démarrer la lecture. Ce bit revient à 0 automatiquement tout de suite après le transfert de la donnée vers EEDATA, (moins d'un cycle)
4) Traiter la donnée disponible dans EEDATA
5) Recommencer au point 2 si on a d'autres données à lire,

IV.6.2 Procédure d'écriture dans l'EEPROM

Pour écrire une donnée dans une position de la mémoire EEPROM, on place l'adresse dans le registre EEADR, la donnée dans le registre EEDATA, on lance l'opération d'écriture à l'aide du bit WR de EECON1 et du registre EECON2. La donnée présente dans EEDATA est alors copiée dans la EEPROM mais cette opération prend 10 ms. A la fin de l'écriture le bit WR revient à zéro automatiquement, le drapeau EEIF est levé ce qui peut déclencher l'interruption EEI si elle a été validée auparavant :

1) Mettre le bit EEPGD à 0 pour pointer sur l'EEPROM de donnée
2) Positionner le bit WREN pour valider l'écriture dans l'EEPROM
3) Interdire les interruptions (si elles ont été validées avant : bit INTCON.GIE)

4) Placer l'adresse relative de la position à écrire dans EEADR (L'adresse de la première position est 0)
5) Placer la donnée à écrire dans le registre EEDATA
6) - Ecrire 55h dans EECON2 (commande de process hardwares)
   - Ecrire AAh dans EECON2 (commandes de process hardwares)
   - Positionner le bit WR pour démarrer l'opération d'écriture, ce bit revient automatiquement à 0 à la fin de l'écriture (après 10 ms)
7) attendre la fin de l'écriture en surveillant le bit WR ou le drapeau EEIF
8) Recommencer au point (4) si on a d'autres données à écrire,
9) Autoriser les interruptions (si besoin)
10) Remettre à Zéro le bit WREN (si besoin)

Les bits WREN et WR ne peuvent être positionné dans la même instruction. WR ne peut être positionné que si le bit WREN a été positionné avant.

IV.7 La mémoire Programme ou mémoire flash

Cette mémoire de 8 x 1024 mots de 14 bits sert à stocker le programme, mais elle est accessible par programme et peut donc être utilisée comme une extension de la mémoire EEPROM de données. Elle est non volatile (flash) et reprogrammable à souhait. Chaque position de 14 bits contient une instruction. L'emplacement du programme peut se situer à n'importe quel endroit de la mémoire. Cependant il faut savoir que suite à un RESET ou lors de la mise sous tension, le PIC commence l'exécution à l'adresse 0000_H. De plus, lorsqu'il y a une interruption, le PIC va à l'adresse 0004_H. Il est donc nécessaire de bien organiser le programme si celui-ci utilise des interruptions.

Le programme exécutable par le PIC est implanté dans la mémoire flash à l'aide d'un programmateur (hard+soft) sur lequel nous reviendrons dans la suite de ce document.

L'accès à la mémoire flash par les instructions du programme se fait de la même façon que l'accès à l'EEPROM de données, à l'exception des points suivant :

- Le bit EEPGD doit être placé à 1

- Le registre EEADR (8 bits) seul ne suffit pas à adresser les 8k de mémoire programme, on lui accole le registre EEADRH dans lequel il faut écrire la partie haute de l'adresse. On obtient ainsi les 13 bits nécessaires pour adresser 8K.
- Le registre EEDATA (8 bits) seul ne suffit pas pour contenir les 14 bits contenus dans une position de la mémoire programme. On lui accole le registre EEDATH qui contiendra les 6 bits supérieurs.
- Il faut insérer deux instructions NOP dans les cycles de lecture/écriture. Pendant la phase d'écriture, le processeur arrête l'exécution des instructions. Il n'est donc pas nécessaire d'attendre la fin de l'écriture pour continuer car ceci se fait automatiquement, à la fin de l'écriture, l'exécution reprend à l'instruction qui suit le 2ème NOP,

IV.7.1 Procédure lecture dans la mémoire programme

1) Mettre le bit EEPGD à 1 pour pointer sur la mémoire programme
2) Placer l'adresse de la position à lire dans EEADRH:EEADR
3) Mettre le bit RD à 1 pour démarrer la lecture
4) Attendre 2 cycles machine (2 instructions NOP)
5) Lire le contenu des registres EEDATH:EEDATA
6) Recommencer au point 2) si on a d'autres données à lire

IV.7.2 Procédure d'écriture dan la mémoire programme

1) Interdire les interruptions (si elles ont été validées avant)
2) Mettre le bit EEPGD à 1 pour pointer sur la mémoire programme
3) Positionner le bit WREN pour valider l'écriture dans la mémoire programme
4) Placer l'adresse de la position à écrire dans EEADRH:EEADR
5) Placer la donnée à écrire dans le registre EEDATH:EEDATA
6) - Ecrire 55h dans EECON2 (commande de process hardwares)
   - Ecrire AAh dans EECON2 (commandes de process hardwares)
   - Positionner le bit WR pour démarrer l'opération d'écriture, ce bit revient automatiquement à 0 à la fin de l'écriture (après 10 ms)
7) Exécuter 2 instructions NOP
8) Recommencer au point 4 si on a d'autres donnée à écrire
9) Autoriser les interruptions (si besoin)
10) Remettre à Zéro le bit WREN (si besoin)

Remarque : après le point (6), ça ne sert à rien de scruter le bit WR pour attendre la fin de l'écriture. Le processeur arrête l'exécution du programme pendant la phase d'écriture dans la mémoire programme. La boucle de scrutation ne sera exécutée qu'après la fin de l'écriture ce qui la rend tout à fait inutile. Il suffit de mettre deux instructions NOP avant de continuer.

Exercice 8)

Programme qui écrit l'alphabet majuscule dans la mémoire EEPROM de données.

Exercice 9) EEPROM-D vers RAM

Programme qui utilise la directive DE pour initialiser les premières positions de l'EEPROM de données avec la chaîne "BONJOUR CHER AMI". Le programme doit ensuite lire ces caractères (1 par 1) dans l' EEPROM et les copier dans la RAM à partir de la position 110h

Exercice 10) Mem-Prog vers RAM

Programme qui lit 20 positions de la mémoire programme débutant à la position 250 (FAh) et les copie dans la RAM à partir de la position 110h

IV.8 Les interruptions

Une interruption provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption. A la fin de cette procédure, le microcontrôleur reprend le programme principal à l'endroit où il l'a laissé. A chaque interruption sont associés deux bits, un bit de validation et un drapeau. Le premier permet d'autoriser ou non l'interruption, le second permet au programmeur de savoir de quelle interruption il s'agit.

Sur le 16F876/877, l'es interruptions sont classées en deux catégories, les interruptions primaires et les interruptions périphériques. Elles sont gérées par les registres :

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIE1 (bk1)	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIR1 (bk0)	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE2 (bk0)	-	-	-	EEIE	BCLIE	-	-	CCP2IE
PIR2 (b1)	-	-	-	EEIF	BCLIF	-	-	CCP2IF
OPTION_REG		INTEDG						

- Toutes les interruptions peuvent être validées/interdites par le bit INTCON.GIE
- Toutes les interruptions périphériques peuvent être validées/interdites par le bit INTCON.PEIE
- Chaque interruption peut être validée/interdite par son bit de validation individuel

En résumé, pour valider une interruption périphérique (par exemple), il faut positionner 3 bits, GIE, PEIE et le bit individuel de l'interruption.

IV.8.1 Déroulement d'une interruption

Lorsque l'événement déclencheur d'une interruption intervient, alors son drapeau est positionné à 1 (levé). Si l'interruption a été validée (bits de validations = 1), elle est alors déclenchée : le programme arrête ce qu'il est en train de faire et va exécuter la procédure d'interruption qui se trouve à l'adresse 4 en exécutant les étapes suivantes :

- l'adresse contenue dans le PC (Program Counter) est sauvegardée dans la pile, puis remplacée par la valeur 0004 (adresse de la routine d'interruption).
- Le bit GIE est placé "0" pour inhiber toutes les interruptions (afin que le PIC ne soit pas dérangé pendant l'exécution de la procédure d'interruption).
- A la fin de la procédure d'interruption (instruction RETFIE) :
- le bit GIE est remis à 1 (autorisant ainsi un autre événement)
- le contenu du PC est rechargé à partir de la pile ce qui permet au programme de reprendre là où il s'est arrêté

Deux remarques importantes sont à faire :

Le drapeau reste à l'état haut même après le traitement de l'interruption. Par conséquent, il faut toujours le remettre à "0" à la fin de la routine d'interruption sinon l'interruption sera déclenchée de nouveau juste après l'instruction RETFIE

Seul le PC est empilé automatiquement. Si cela est nécessaire, les registres W et STATUS doivent être sauvegardés en RAM puis restaurés à la fin de la routine pour que le microcontrôleur puisse reprendre le programme principal dans les mêmes conditions où il l'a laissé.

IV.8.2 Les sources d'interruption

Source d'interruption	Validation	Flag	PEIE
Débordement Timer 0	INTCON.T0IE	INTCON.T0IF	non
Front sur RB0/INT	INTCON.INTE	INTCON.INTF	non
Front sur RB4-RB7	INTCON.RBIE	INTCON.RBIF	non
Fin de conversion A/N	PIE1.ADIE	PIR1.ADIF	oui
Un Octet est reçu sur l'USART	PIE1.RCIE	PIR1.RCIF	oui
Fin transmission d'un octet sur l'USART	PIE1.TXIE	PIR1.TXIF	oui
Caractère émis/reçu sur port série synchrone	PIE1.SSPIE	PIR1.SSPIF	oui
Débordement de Timer 1	PIE1.TMR1IE	PIR1.TMR1IF	oui
Timer 2 a atteint la valeur programmée	PIE1.TMR2IE	PIR1.TMR2IF	oui
Lecture/écriture terminée sur Port parallèle (16F877)	PIE1.PSPIE	PIR1.PSPIF	oui
Capture/comparaison de TMR1 avec module CCP1	PIE1.CCP1IE	PIR1.CCP1IF	oui
Capture/comparaison de TMR1 avec module CCP2	PIE2.CCP2IE	PIR2.CCP2IF	oui
Fin d'écriture en EEPROM	PIE2.EEIE	PIR2.EEIF	oui
Collision de bus SSP en mode I2C	PIE2.BCLIE	PIR2.BCLIF	oui

IV.8.3 L'interruption INT (Entrée RB0 du port B)

Cette interruption est provoquée par un changement d'état sur l'entrée RB0 du port B quand elle est programmée en entrée. En plus de son bit de validation INTE et son drapeau INTF, elle est gérée aussi par le bits INTEDG (OPTION_REG) qui détermine le front sur lequel l'interruption se déclenche, 1=montant, 0=descendant

IV.8.4 L'interruption RBI (RB4 A RB7 du port B)

Cette interruption est provoquée par un changement d'état sur l'une des entrées RB4 à RB7 du port B, Le front n'a pas d'importance. Les bits associés sont RBIE (validation) et RBIF (drapeau)

IV.8.5 Les autres interruptions

Les autres interruptions seront abordées au moment de l'étude des modules qui les déclenchent.

IV.9 Le Timer TMR0

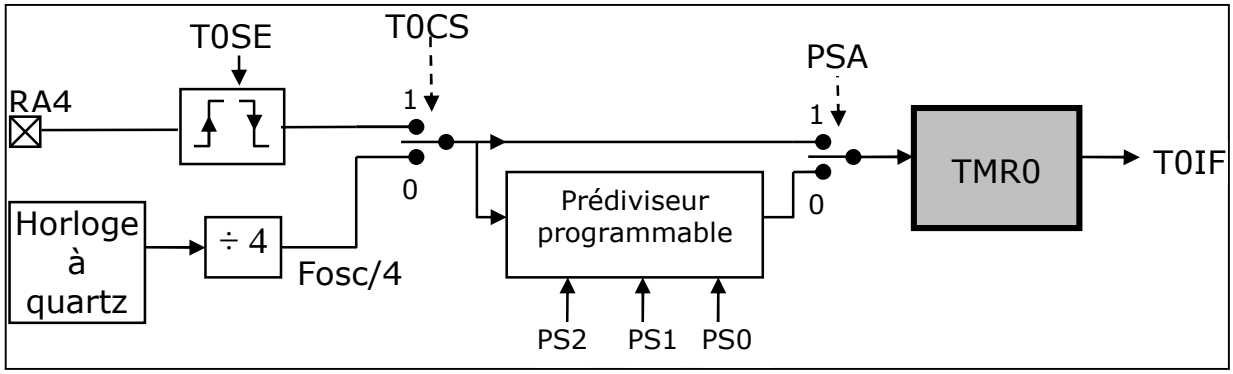
C'est un compteur 8 bits ayant les caractéristiques suivantes :

- Il est incrémenté en permanence soit par l'horloge interne Fosc/4 (mode timer) soit par une horloge externe appliquée à la broche RA4 du port A (mode compteur). Le choix de l'horloge se fait à l'aide du bit T0CS du registre OPTION_REG
  - T0CS = 0 → horloge interne
  - T0CS = 1 → horloge externe appliquée à RA4
- Dans le cas de l'horloge externe, Le bit T0SE du registre OPTION_REG permet de choisir le front sur lequel le TIMER s'incrémente.
  - T0SE = 0 → incrémentation sur fronts montants
  - T0SE = 1 → incrémentation sur fronts descendants
- Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport est fixés par les bits PS0, PS1 et PS2 du registre OPTION_REG (tableau ci-contre). L'affectation ou non du prédiviseur se fait à l'aide du bit PSA du registre OPTION_REG
  - PSA = 0 → on utilise le prédiviseur
  - PSA = 1 → pas de prédiviseur (affecté au chien de garde)

PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

- Le contenu du timer TMR0 est accessible par le registre qui porte le même nom. Il peut être lu ou écrit à n'importe quel moment. Après une écriture, l'incrémentation est inhibée pendant deux cycles instruction
- Au débordement de TMR0 (FF → 00), le drapeau T0IF est placé à 1. Ceci peut déclencher l'interruption T0I si celle-ci est validée

OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0



Voici quelques valeurs à titre indicatif

Tempo désirée	Prescaler	Variable	Tempo obtenue	erreur
0.1 s	2	195	99840 µs	160 µs
0.25 s	15	61	249856 µs	144 µs
0.5 s	32	61	499712 µs	288 µs
0.5 s	256	8	524288 µs	24288 µs
1 s	64	61	999424 µs	576 µs

Exercice 11) Clignoter LED / TMR0

Clignoter une LED branchée sur RB0, delay voisin de 0.5s à l'aide de TMR0

a) Par scrutation du drapeau T0IF (pas d'interruption)
b) En utilisant l'interruption T0I

IV.10 Le Watchdog Timer WDT (Chien de garde)

C'est un compteur 8 bits incrémenté en permanence (même si le µC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester *planté* en cas de blocage du microcontrôleur par un processus indésirable non contrôlé
- Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations

L'horloge du WDT a une période voisine de 70 µs ce donne un Time-Out toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passer le signal Time-Out dans un prédiviseur programmable (partagé avec le timer TMR0). L'affectation se fait à l'aide du bit PSA du registre OPTION_REG

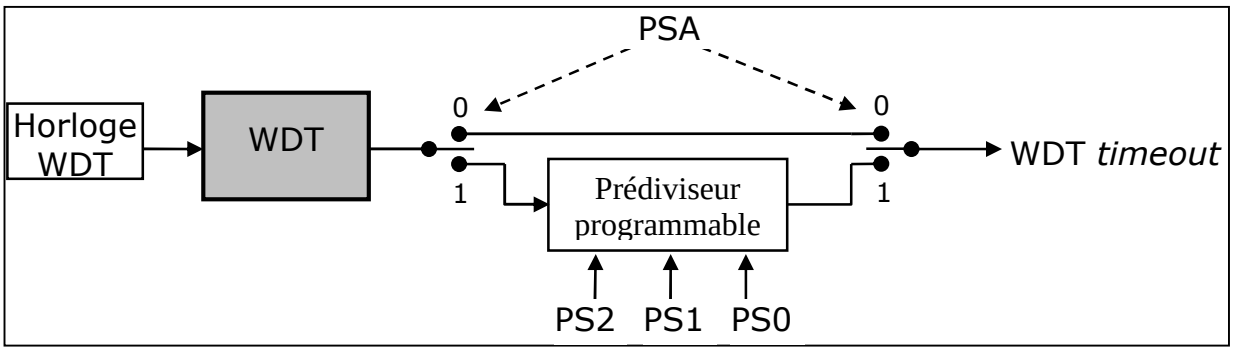
- PSA = 1 → on utilise le prédiviseur
- PSA = 0 → pas de prédiviseur (affecté à TMR0)

Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (voir tableau ci-contre)

PS2	PS1	PS0	Div
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

L'utilisation du WDT doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un WDT timeOut lors de l'exécution d'un programme, on a deux possibilités :

- Inhiber le WDT d'une façon permanente en mettant à 0 le bit WDTE dans l'EEPROM de configuration
- Remettre le WDT à 0 périodiquement dans le programme à l'aide de l'instruction CLRWDT pour éviter qu'il ne déborde



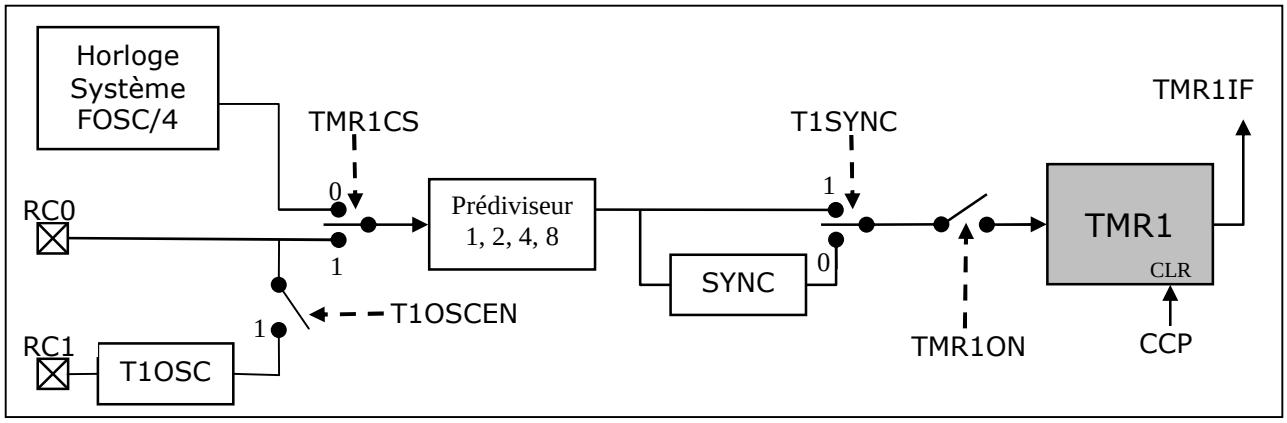
Voici quelques valeurs à titre indicatif

Tempo désirée	Prescaler	Variable	Tempo obtenue	erreur
0.1 s	-	6	108 ms	8 ms
0.25 s	-	14	252 ms	2 ms
0.5 s	-	28	504 ms	4 ms
0.5 s	32	-	576 ms	76 ms
1 s	-	56	1008 ms	8 ms

IV.11 Le Timer TMR1

TMR1 est un Timer/Compteur 16 bits accessible en lecture/écriture par l'intermédiaire des registres 8 bits TMR1H (bank0) et TMR1L (bank0) qui constituent sa partie haute et sa partie basse.

On le configure à l'aide du registre T1CON (bank0)



- A son débordement (FFFFh → 0000h) le drapeau PIR1.TMR1IF (bank0) est positionné ce qui peut déclencher l'interruption périphérique TMR1I si elle est validée par son bit de validation PIE1.TMR1IE (bank1).
- TMR1 peut fonctionner dans 3 modes différents :
  - Timer Synchrone (horloge interne)
  - Compteur Synchrone (horloge externe)
  - Compteur Asynchrone (horloge externe)

  Le bit de contrôle TMR1CS détermine le fonctionnement en Timer ou en Compteur et le bit T1SYNC détermine le mode de fonctionnement en synchrone ou en asynchrone

- TMR1 peut être arrêté/démarré à l'aide du bit TMR1ON
- TMR1 peut être RAZ à l'aide du module de capture/comparaison CCP
- TMR1 peut être précédé d'un prédiviseur (Prescaler) qui peut diviser la fréquence par 1, 2, 4 ou 8 selon la valeur des bits T1CKPS1 et T1CKPS0

IV.11.1 Le mode Timer

Dans ce mode, TMR1 est incrémenté par l'horloge système Fosc/4 éventuellement prédivisée. Le bit de synchronisation n'a pas d'effet car

l'horloge Fosc/4 est toujours synchronisée sur l'horloge système.

IV.11.2 Le mode Compteur

Dans ce mode, TMR1 est incrémenté à chaque front montant de l'horloge externe T1CKI (RC0) ou l'horloge interne générée par l'oscillateur dédié T1OSC à condition de positionner le bit T1OSCEN à 1 et de brancher un quartz entre les broche RC0 et RC1.

En mode compteur, RC0 et RC1 sont automatiquement configurées en entrée, on n'a pas besoin de configurer les bits TRISC,0 et TRISC,1

- En fonctionnement **Synchrone**, l'horloge externe (éventuellement prédivisée) n'incrémente pas directement le timer mais elle est synchronisée sur l'horloge système ce qui peut entraîner un délai de l'ordre de 1 cycle machine. Dans cette configuration

- En fonctionnement **Asynchrone**, l'horloge externe (éventuellement prédivisée) incrémente le timer indépendamment de l'horloge système.

En mode Compteur Asynchrone, on ne peut pas utiliser les modules CCP1 et CCP2 pour faire des captures ou des comparaisons sur TMR1

IV.11.3 Le registre de control de T1CON

—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

T1CKPS1,T1CKPS0 : Control du prescaler
- 00 : division par 1
- 01 : division par 2
- 10 : division par 4
- 11 : division par 8

T1OSCEN : Validation de l'Oscillateur associé à TMR1
- 0 : Oscillateur arrêté
- 1 : Oscillateur activé

T1SYNC : Synchronisation de l'horloge externe (ignoré en mode timer)
- 0 : Synchronisation
- 1 : pas de synchronisation

TMR1CS : Choix de l'horloge du Timer
- 0 : horloge système (Fosc/4) : mode timer
- 1 : Horloge externe : mode compteur

TMR1ON : Démarrer arrêter le timer
- 0 : Timer stoppé
- 1 : Timer en fonctionnement

Voici quelques valeurs à titre indicatif

Tempo désirée	Prescaler	Variable	Tempo obtenue	erreur
0.5 s	8	-	524888 µs	24288 µs
1 s	1	15	983040 µs	16860 µs
5 s	4	19	4980736 µs	19364 µs
10 s	1	153	10027008	27008 µs

Exercice 12) Clignoter LED / interruption TMR1

Clignoter une LED reliée à RD0. La temporisation voisine de 0.5s est réalisée à l'aide de TMR1 est son interruption TMR1I

Exercice 13) Clignoter LED / scrutation de TMR1

Clignoter une LED reliée à RE0. La temporisation voisine de 0.5s est réalisée à l'aide de TMR1 par scrutation du drapeau TMR1IF

IV.12 Les module de Comparaison/Capture CCP1 et CCP2

Chacun des modules CCP1 et CCP2 permet :

- Soit de CAPTURER en un seul coup le contenu du double registre TMR1
- Soit de COMPARER en permanence son contenu avec un registre 16 bits et de déclencher un événement au moment de l'égalité.

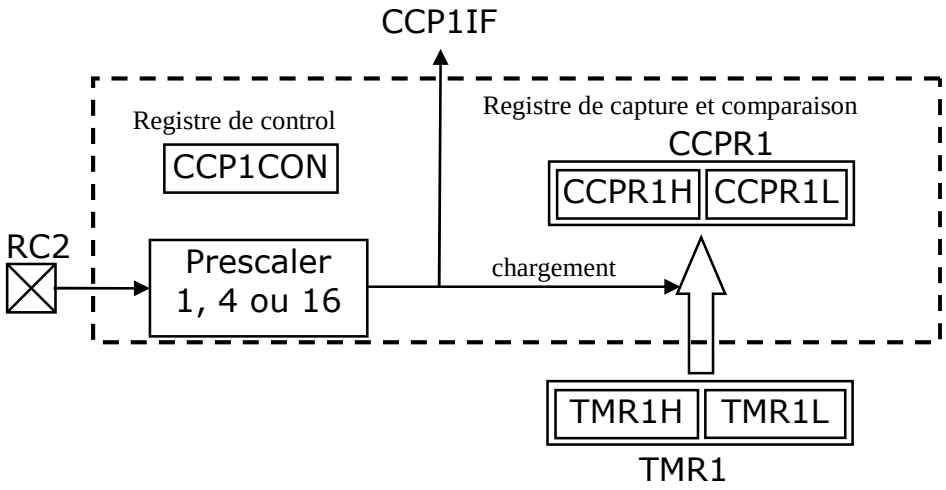
Ces modules ne fonctionnent pas si TMR1 est configuré en mode Compteur non synchronisé

IV.12.1 Le module CCP1

Ce module est constitué de :

- Un registre 16 bits CCPR1 utilisé pour la capture ou la comparaison de TMR1. Il est accessible par sa partie basse CCP1L et sa partie haute CCP1H
- Un registre de contrôle 8 bits CCP1CON.
- Un prédiviseur permettant de filtrer les événements déclencheurs de capture venant de la broche RC2

IV.12.1.1 Le mode Capture



Dans ce mode le contenu de TMR1 est copié dans CCPR1 chaque fois qu'un événement intervient sur la broche RC2. Le choix de l'événement déclencheur se fait en programmant le prescaler à l'aide des bits 0 à 3 du registre de contrôle CCP1CON. On a le choix parmi les événements suivants :

- A chaque front descendant
- A chaque front montant (prescaler 1:1)
- A chaque 4ème front montant (prescaler 1:4)
- A chaque 16ème front montant (prescaler 1:16)

A la fin de la capture, le drapeau CCP1IF est positionné, l'interruption périphérique associée est déclenchée si elle a été validée.

Plusieurs aspects sont à noter comme :

- Si on veut déclencher la capture par un signal externe, celui-ci doit être appliqué sur la proche RC2 qui doit être configurée en entrée par le bit TRISC,2
- Si on veut déclencher la capture par programme en changeant la valeur du bit RC2, celui-ci doit être configuré en sortie par le bit TRISC,2
- Lors de la modification du mode de capture, une interruption indésirable peut intervenir. L'utilisateur doit veiller à masquer l'interruption CCP1I avant de procéder à cette modification et de baisser le drapeau CCP1IF après la modification.
- En mode Sleep, le prescaler et le drapeau CCP1IF restent opérationnels. le positionnement du drapeau peut déclencher un Wake-up si le bit de validation CCP1IE a été validé auparavant

IV.12.1.2 Le registre de configuration CCP1CON

—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

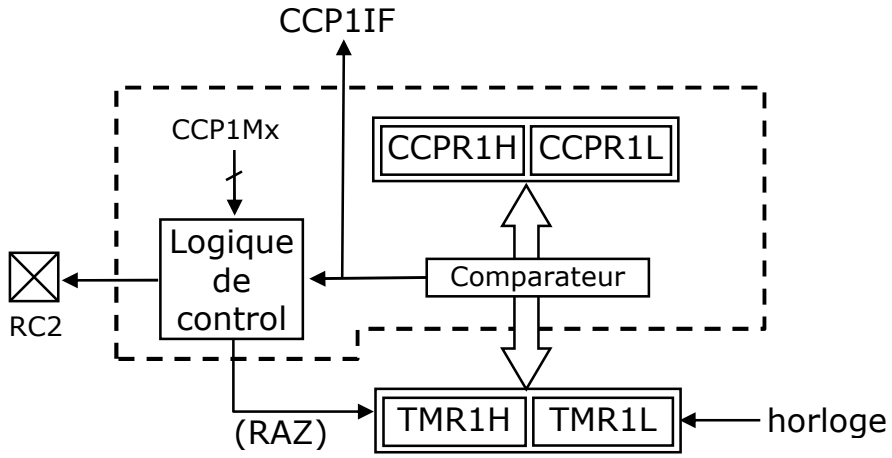
DC1B1, DC1B0 : utilisés en PWM mode

Ce sont les 2 bits de poids faible des 10 bits MWM duty cycle. Les 8 autres bits sont dans le registre CCPR1L

CCP1M3 à CCP1M0 : mode de fonctionnement du module CCP1

0000 : Module arrêté (reset module)
0100 : Capture sur chaque front descendant
0101 : Capture sur chaque front montant
0110 : Capture tous les 4 fronts montants
0111 : capture tous les 16 fronts montants
1000 : Mode comparaison (interruption + broche RC2 $0 \rightarrow 1$)
1001 : Mode comparaison (interruption + broche RC2 $1 \rightarrow 0$)
1010 : Mode comparaison (interruption seulement)
1011 : Mode comparaison (déclenche événement spécial)
10xx : PWM mode (modulation de largeur d'impulsion), ce mode ne sera pas traité dan ce cours

IV.12.1.3 Le mode Comparaison



Dans ce mode le registre CCPR1 est comparé en permanence à TMR1. Quand l'égalité intervient, le drapeau CCP1IF passe à 1 et différentes actions sont accomplies selon le mode défini par les bits de configuration CCP1M3:CCP1M0

- **mode 1000 :**
  Au moment de l'égalité, le drapeau CCP1IF est le bit RC2 passent à 1. C'est à l'utilisateur de les remettre à 0 pour une prochaine utilisation. RC2 doit être configuré en sortie.

- **mode 1001 :**
  Au moment de l'égalité, le drapeau CCP1IF passe à 1 et le bit RC2 passe à 0. C'est à l'utilisateur de les remettre à leur état d'origine pour une prochaine utilisation. RC2 doit être configuré en sortie.

- **mode 1010 :**
  A l'égalité le drapeau CCP1IF passe à 1. La broche RC2 n'est pas utilisée.

- **mode 1011 :**
  A l'égalité, le drapeau CCP1IF passe à 1 et le timer TMR1 est remis à 0

IV.12.2 Le module CCP2

Le module CCP2 est identique au module CCP1, il suffit d'interchanger 1 et 2 et de constater les points suivants :

- CCP2 est associé à la broche RC1
- Il est géré par le registre de control CCP2CON
- Son registre de capture/comparaison est CCPR2 = CCPR2H:CCPR2L
- En mode comparaison 1011, A l'égalité :
  - o le drapeau CCP2IF passe à 1
  - o RAZ de TMR1
  - o envoi d'un GO vers le convertisseur analogique numérique

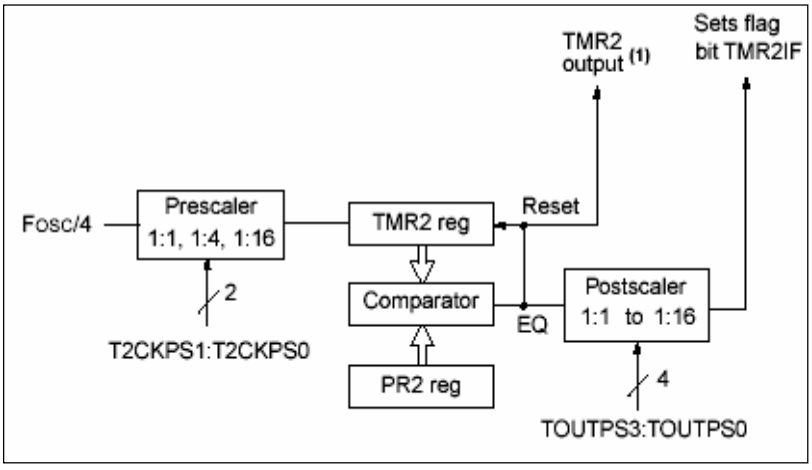
Exercice 14) générer signal TMR1/CCP1

Programme qui génère le signal ci-dessous sur la sortie RE0 en utilisant TMR1 associé à CCP1

IV.13 Le Timer TMR2

TMR2 est un timer 8 bits accessible en lecture écriture constitué de :

- un registre de control T2CON
- un prédiviseur (1,4,16)
- un registre de période PR2 accessible en lecture/écriture
- un comparateur,
- un postdiviseur (1 à 16)



- TMR2 est incrémenté par l'horloge interne Fosc/4. Quant il atteint la valeur du registre PR2, le comparateur génère un signal qui :
  - Remet TMR2 à 0
  - incrémente le postscaler
- Au débordement du postscaler, le drapeau TMR2IF est positionné, l'interruption correspondante et déclenchée si elle est validée
- TMR2 est remis à zéro à chaque RESET et à chaque écriture dans PR2
- Le prescaler et le postscaler sont initialisés à chaque écriture dans TMR2 ou dans T2CON et au RESET du processeur
- Le fonctionnement de TMR2 est configuré à l'aide du registre de control T2CON :

—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

TOUTPS3:TOUTPS0 : ratio du postscaler
0000 : division par 1
0001 : division par 2
. . .
1111 : division par 16

TMR2ON : démarrer arrêter TMR2
0 : TMR2 off
1 : TMR2 on

T2CKPS1,T2CKPS0 : ratio du prescaler
00 : prédiviseur par 1
01 : prédiviseur par 4
1x : prédiviseur par 16

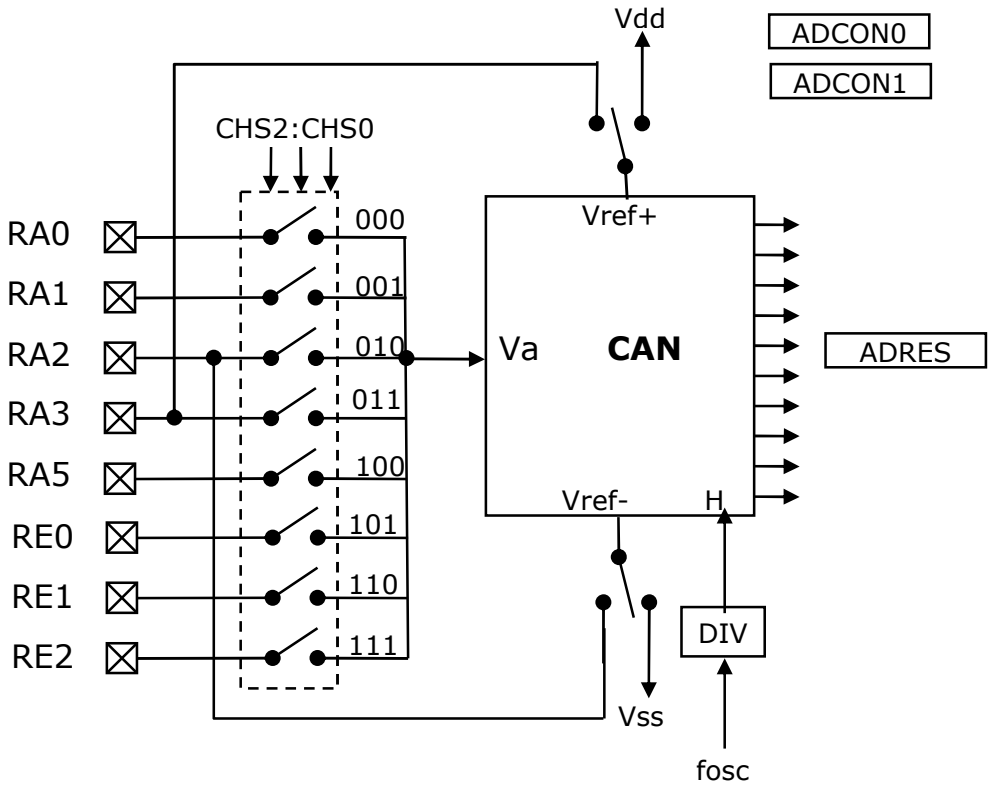
Voici quelques valeurs à titre indicatif

Tempo désirée	Prescaler	PR2	Postscaler	Variable	erreur
0.1 s	16	250	5	5	0
0.25 s	16	125	5	25	0
0.5 s	16	250	5	25	0
1 s	16	250	10	25	0

Exercice 15) Clignoter LED / scrutation de TMR2

Clignoter une LED reliée à RE0. La temporisation voisine de 0.5s est réalisée à l'aide de TMR2 par scrutation du drapeau TMR2IF

IV.14 Le module de conversion A/N



Ce module est constitué d'un convertisseur Analogique Numérique 10 bits dont l'entrée analogique peut être connectée sur l'une des 8 (5 pour 16F876) entrées analogiques externes. On dit qu'on a un CAN à 8 canaux. Les entrées analogiques doivent être configurées en entrée à l'aide des registres TRISA et/ou TRISE.

Les tensions de références permettant de fixer la dynamique du convertisseur. Elles peuvent être choisies parmi Vdd, Vss, Vr+ ou Vr-

La conversion démarre quand on place le bit GO/DONE à 1. Ceci peut être fait par le programme utilisateur ou automatiquement par le module CCP2 s'il est configuré en mode comparaison avec événement spécial.

A la fin de la conversion, le résultat de conversion est recopié dans les registres ADRESH et ADRESL, le bit GO/DONE repasse automatiquement à 0 et le drapeau ADIF *(situé dans PIR1)* passe à 1 ce qui peut déclencher l'interruption associée si elle est validée

Le control du module se fait par les deux registres ADCON0 et ADCON1

ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON

ADCS1:ADCS0 : Choix de l'horloge de conversion donc du temps de conversion (voir paragraphe IV.14.2)
00 : Fosc/2
01 : Fosc/8
10 : Fosc/32
11 : Oscillateur RC dédié au CAN

CHS2:CHS0 : choix de l'entrée analogique
000 = channel 0, (RA0)
001 = channel 1, (RA1)
010 = channel 2, (RA2)
011 = channel 3, (RA3)
100 = channel 4, (RA5)
101 = channel 5, (RE0)
110 = channel 6, (RE1)
111 = channel 7, (RE2)

GO/DONE : Une conversion démarre quand on place ce bit à 1. A la fin de la conversion, il est remis automatiquement à zéro. Ce bit peut aussi être positionné automatiquement par le module CCP2.

ADON : Ce bit permet de mettre le module AN en service

ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0

ADFM : justification à droite ou à gauche du résultat
`1 : justifié à droite 000000XX XXXXXXXX`
`0 : justifié à gauche XXXXXXXX XX000000`

PCFG3:PCFG0 : configuration des E/S et des tensions de références. Les 5 broches de PORTA et les 3 de PORTE peuvent être configurés soit en E/S digitales, soit en entrées analogiques. RA2 et RA3 peuvent aussi être configurées en entrée de référence.

PCFG3: PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	A/R/D
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0/0
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	7/1/0
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0/3
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/1/3
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0/5
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}	2/1/5
011x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0/8
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	6/2/0
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0/2
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	5/1/2
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	4/2/2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/2/3
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/2/4
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0/7
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2	1/2/5

IV.14.1 Temps d'acquisition

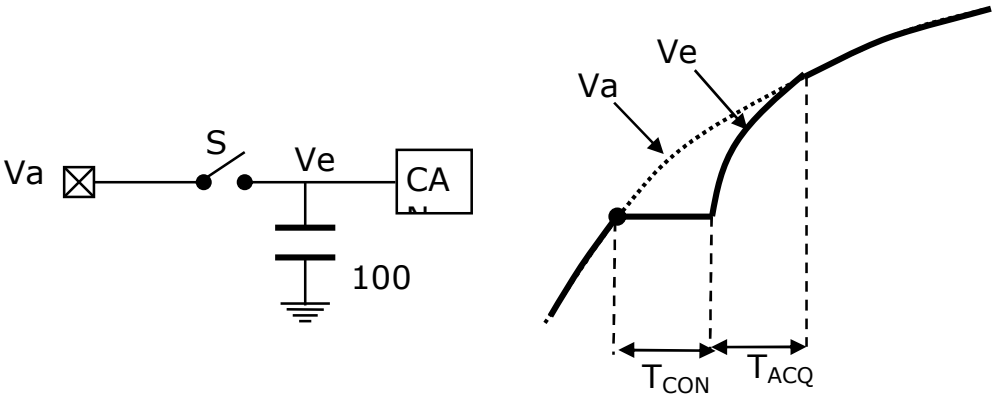
Pendant la conversion, la tension Ve à l'entrée du convertisseur A/N doit être maintenue constante. Le PIC dispose d'un échantillonneur bloqueur intégré constitué d'un interrupteur S et d'une capacité de maintien de C=100

pF. Pendant le temps de conversion, S est maintenu ouvert, la capacité bloque Ve à une valeur constante. A la fin de la conversion, S se ferme, la tension Ve rejoint la tension analogique d'entrée Va au bout d'un temps d'acquisition qui dépend de la constante de temps RC, R étant la somme de la résistance d'interconnexion (Ric), la résistance du sampling switch S (Rss) et la résistance de la source de tension Va (Rs).

- La valeur de Rs ne doit pas dépasser 10 kΩ.
- La valeur Ric est ≤ 1 kΩ
- La valeur de Rss dépend de la tension d'alimentation, elle est de l'ordre de 7kΩ pour Vdd=5V et de 5 kΩ pour Vdd = 6V

T_{ACQ} : Temps d'acquisition = 2 µs + Tc + CT

Tc : temps de charge du condenseur = (Ric+Rss+Rs) C Ln(2047)
CT : Coefficient de température = (Tp -25°C) 0.05 µs/°C
(Tp = Température Processeur)



Avec Ric = 1k, Rss = 7k, Rs = 2k, Tp = 50 °C :

Tc = 10k x 100pF x Ln(2047) = 7,6 µs
CT = 25 x 0.05 µs = 1,25 µs

$$T_{ACQ} = 2 + 7,6 + 1,25 \text{ µs} = 10, 85 \text{ µs}$$

L'acquisition commence :

- Après la fin d'une conversion
- au moment du choix d'un canal si convertisseur validé (ADON=1)
- au moment de validation du convertisseur si canal choisi

IV.14.2 Temps de conversion

Le temps de conversion est égal à (12 +2) T_{AD}
T_{AD} est le temps de conversion d'un bit, il dépend de l'horloge système et du prédiviseur (div) choisi. Les choix doivent être ajusté pour que que T_{AD} ne dépasse pas 1,6 µs

\Quartz Div \	20Mhz	5Mhz	4Mhz	2Mhz
2	0,1 µs	0,4 µs	0,5 µs	1 µs
8	0,4 µs	1,6 µs	2 µs	4 µs
32	1,6 µs	6,4 µs	8 µs	16 µs

Tableau IV.1 : Temps de conversion d'un bit TAD

Une conversion démarre au moment du positionnement du bit GO_DONE, l'interrupteur S est ouvert et la conversion est réalisée en 12 T_{AD}. A la fin, le bit GO_DONE est remis à 0 et le drapeau ADIF est placé à 1. Le module attend 2 T_{AD} supplémentaires avant de fermer l'interrupteur S et commencer une nouvelle acquisition.

IV.14.3 Fréquence d'échantillonnage

Si on veut échantillonner un signal variable, La période d'échantillonnage Te doit être supérieur ou égale à $T_{emin} = T_{ACQ} + T_{CONV}$

IV.14.4 Valeur numérique obtenue

Quelle est la relation entre la tension analogique convertie et le nombre N recueilli dans le registre ADRES ?

Si on note :

Q = pas de quantification = $(V_{ref+} - V_{ref-})/1024$
Va = tension analogique à convertir
N = valeur numérique obtenue,

$$N = \text{valeur entière de } (Va - V_{ref-}) / Q$$

Avec Vref- = masse, on obtient

$$N = \text{int } (Va / Q)$$

exemple :

Vref+ = Vdd = 5V, Vref- = 0, Vin = 4 V
Q = 5V/1024 = 0,0048828125 V
N = 4V / 0,0048828125 = 819

IV.14.5 Programmation

1) Si des entrée de PORTE sont utilisées, placer le bit TRISE,PSPMODE à 0
2) Configurer les E/S en Analogique/digital/Référence (ADCON1)
3) Configurer les entrées analogiques en entrées (TRISA, TRISE)
4) Définir l'horloge de conversion, Valider le module (ADCON0)
5) Choisir le canal à convertir (ADCON0)

6) attendre temps d’acquisition (20 à 25 µs dans le cas général)
7) Lancer la conversion, GO_DONE = 1 (ADCON0)
8) Attendre fin de conversion, GO_DONE = 0 ou interruption si validée
9) Lire le résultat

- Arrêter le convertisseur ou recommencer au point 6

IV.15 L'USART

L'USART (Universal Synchronous Asynchronous Receiver Transmitter) est l'un des deux modules de communication série dont dispose le PIC 16F876/877. L'USART peut être configuré comme système de communication asynchrone full duplex ou comme système synchrone half duplex.

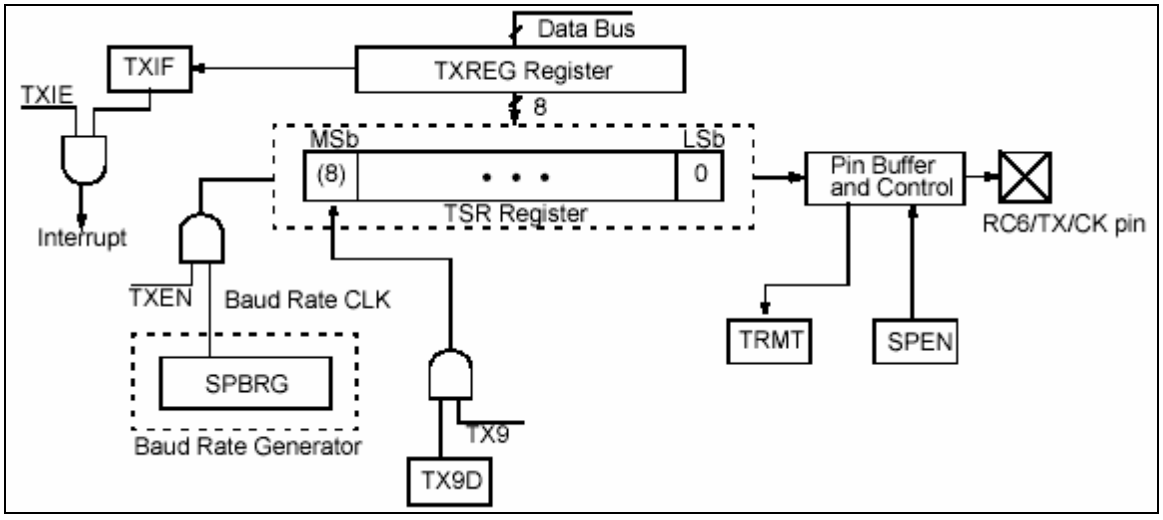
La communication se fait sur les deux broches RC6/TX et RC7/RX qui doivent être configurés toutes les deux en ENTREE par TRISC. (oui, j'ai bien dit toutes les deux)

IV.15.1 Mode Asynchrone

Si on place le bit SYNC du registre TXSTAT à 0, l'USART fonctionne dans le mode asynchrone standard, 10 (ou 11) bits sont transmis ou reçus dans l'ordre ci-dessous :
- 1 bit de START (toujours 0)
- 8 ou 9 bits de donnée (LSB d'abord)
- 1 bits de STOP (toujours 1)

- La configuration du port se fait par les registres TXSTA et RCSTA
- La transmission se fait sur la broche RC6/TX et la réception sur la broche RC7/RX
- La vitesse de communication est déterminée par un générateur de baud rate dédié.
- La parité n'est pas gérée d'une façon matérielle, elle peut être gérée par soft si son utilisation est nécessaire.
- L'accès au port en lecture ou écriture se fait par les registres tampon RCREG et TXREG. La transmission et la réception se font par deux registres à décalage, un pour la transmission (TSR) et un pour la réception (RSR). L'accès au registres tampon peut se faire alors que les registre à décalage sont en train de transmettre/recevoir une donnée.
- Le control du port se fait par les registres d'état et de control TXSTA et RCSTA
- Le drapeau RCIF et TXIF sont très utiles pour gérer la lecture/écriture dans le port. RCIF est positionné quand le port a terminé une réception et TXIF est positionné quand le buffur de transmission TXREG est "vide".

IV.15.2 Le port en transmission



Le contrôle de la transmission se fait par le registre TXSTA

CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D

CSRC : non utilisé en mode asynchrone
TX9 et TX9D : Pour utiliser le mode 9 bits il faut positionner le bit TX9. Le 9ème bit doit être écrit dans TX9D avant d'écrire les 8 bits de données dans TXREG
TXEN : permet de valider ou interdire la transmission
SYNC : 0 → mode asynchrone, 1 → mode synchrone
BRGH : sélectionne le mode haut débit du générateur de baud rate
TRMT : Indicateur de l'activité du registre à décalage de transmission TSR
1 → TSR libre, 0 → TSR en activité

- Quand on écrit un octet D dans le registre TXREG, le drapeau TXIF passe à 0, ensuite, deux situations sont possibles :
- Le registre de transmission TSR n'est pas occupé, alors la donnée D est transférée immédiatement dans TSR qui commence sa transmission bit bar bit et le drapeau TXIF repasse à 1 pour nous dire que nous pouvons de nouveau écrire dans TXREG.
- Le registre de transmission TSR est occupé à transmettre un octet qui lui été donné auparavant. La donnée D attend dans TXREG, et le drapeau TXIF reste à 0 jusqu'à ce que TSR termine de transmettre l'octet précédent. La donnée D est alors transférée dans TSR qui commence sa transmission bit bar bit. Le drapeau TXIF repasse à 1 pour nous dire que nous pouvons de nouveau écrire dans TXREG.

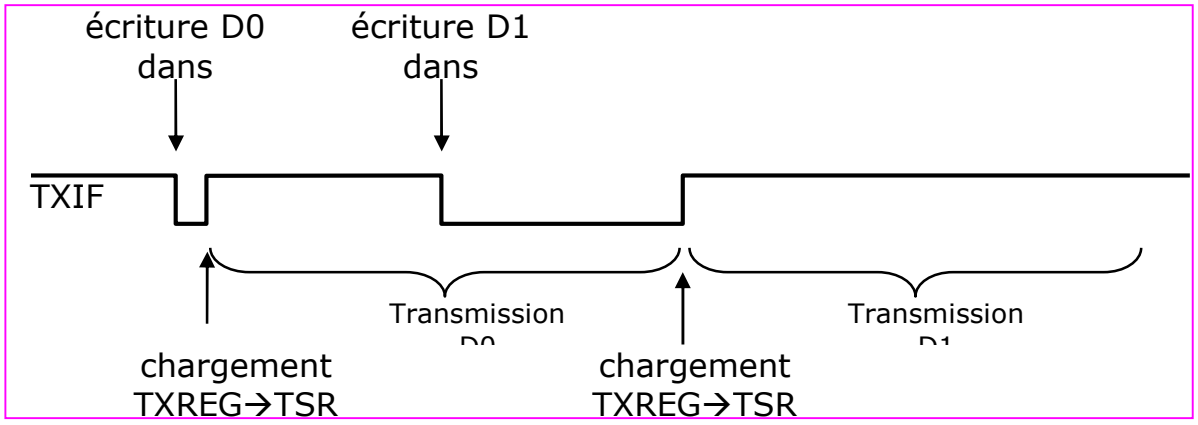


Figure IV.1 : illustration de la transmission de deux octet successifs D_0 et D_1

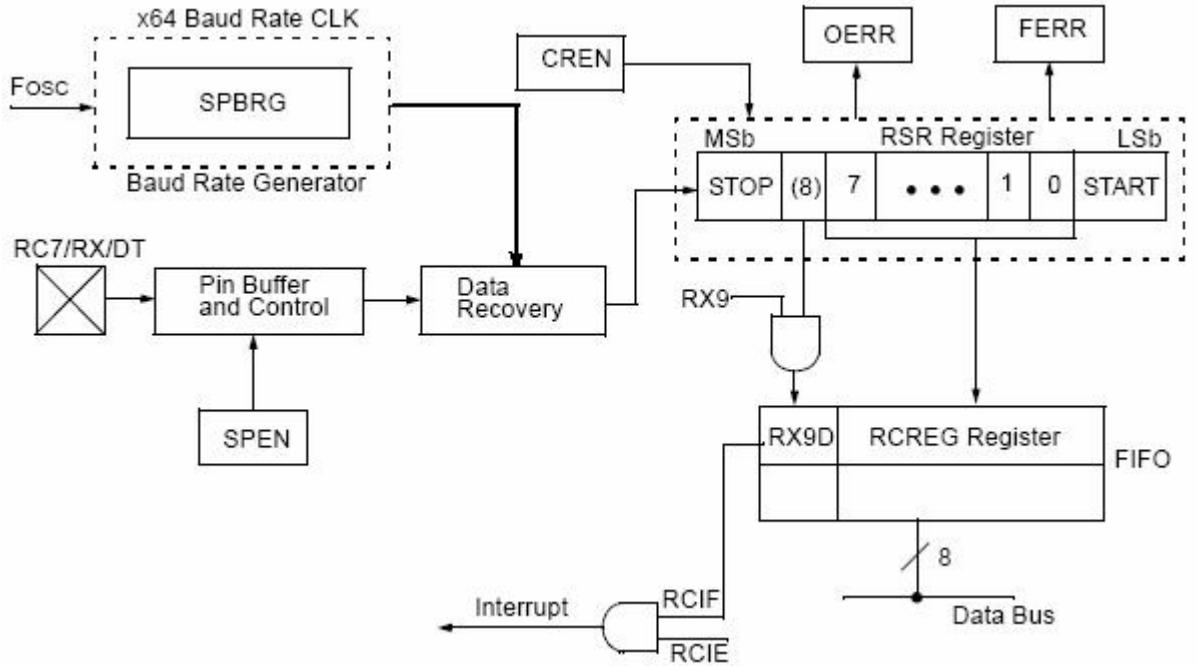
- A la mise sous tension, tous les drapeaux sont à zéro y compris TXIF. Dès qu'on valide la transmission en positionnant le bit TXEN, le drapeau TXIF passe automatiquement à 1, sauf si on écrit d'abord un octet dans TXREG et on valide ensuite le bit TXEN, dans ce cas, dès la validation de TXEN, le contenu de TXREG est transféré dans TSR qui commence sa transmission et le drapeau TXIF passe à 1.

- Comme on vient de le voir, le drapeau TXIF est géré automatiquement, on ne peut pas le modifier directement par programme.

IV.15.3 Les étapes de transmission (sans interruption, mode 8 bits)

1) S'assurer que l'interruption TXI n'est pas validée
2) Configurer la broche TX/RC6 en entrée
3) Configurer le registre TXSTA (mode 8 bits, valider transmission, asynchrone, BRGH)
4) Initialiser le registre SPBRG pour définir la vitesse de transmission
5) Valider le port avec le bit RCSTA.SPEN
6) Vérifier si TXIF=1 c.à.d TXREG est vide
7) Placer la donnée à transmettre dans TXREG
8) recommencer au point 6) tant qu'on a des données à transmettre

IV.15.4 Le port en réception



- La réception est validée par le bit CREN
- La réception d'un octet démarre à la réception du START bit qui commence toujours par une transition $1 \rightarrow 0$
- A la réception du stop bit le contenu du registre à décalage de réception RSR est recopié dans le registre tampon de réception RCREG. Le drapeau RCIF (PIR1.5) est positionné, l'interruption associée est déclenchée si elle est validée. Le drapeau RCIF est remis à zéro automatiquement au moment de la lecture dans RCREG.
- Le registre RCREG est un registre double (FIFO à 2 positions). On peut donc avoir 2 octets en attente dans ce registre et être en train de recevoir un 3ème dans le registre à décalage RSR. A la fin de la réception du 3ème octet, si RCREG est toujours plein, alors le dernier octet reçu est perdu et le bit OERR (Overrun ERRor bit) est positionné ce qui provoque l'arrêt des transfert du registre à décalage RSR vers le buffer RCREG. Pour reprendre

la réception il faut réinitialiser le module de réception en mettant à 0 puis à 1 le bit CREN (↺).

- Le drapeau RCIF ne passe à 0 que quand la pile RCREG est vide
- Si on reçoit un 0 à la position du STOP bit qui doit être toujours à 1, alors le bit FERR (Framing ERRor) est positionné. Ce bit ainsi que le 9ème bit (si utilisé) sont *bufferisés*, Il doivent être lu dans le registre RCSTA avant la lecture du registre RCREG.

Le contrôle de la réception se fait par le registre RCSTA

SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

SPEN : Validation du port série (1 = validé, 0 = Inhibé)

RX9 : Validation du mode 9 bits (1 → mode 9 bits, 0 → mode 8 bits)

SREN : validation de réception d'u seul octet (non utilisé en mode asynchrone)

CREN : Validation du mode réception continue (1 → validé, 0 → inhibé)

ADDEN : validation du mode détection d'adresse en mode 9 bits, utilisé en mode multiprocesseurs (1 → validé, 0 : Inhibé)

FERR : Erreur de synchronisation, lecture seule. (Voir au dessus)

OERR : Erreur débordement du buffer de réception, lecture seule. (Voir au dessus)

RX9D : En mode 9 bits, le 9ème bit est reçu ici

IV.15.5 Les étapes de réception (sans interruption, mode 8 bits)

1) S'assurer que l'interruption RCI n'est pas validée
2) Configurer la broche RX/RC7 en entrée
3) Initialiser le registre SPBRG pour définir la vitesse de communication
4) Configurer le registre TXSTA (asynchrone, BRGH)
5) Configurer le registre RCSTA (validation port, mode 8 bits, valider réception continue)
6) Attendre que drapeau RCIF passe ce qui annonce la fin de réception d'un octet
7) Lire la donnée reçue dans le registre RCREG
8) recommencer au point 6) tant qu'on a des données à recevoir

IV.15.6 La vitesse de communication

La vitesse de communication est déterminée par le générateur de rythme BRG (Baud Rate Generator) qui est dédié au port série. La vitesse de communication est définie à l'aide du registre de control SPBRG et du bit BRGH (TXSTA.2) qui quadruple la vitesse quand il est positionné. Il préférable d'utiliser le mode haute vitesse (BRGH=1) car permet d'obtenir une meilleur précision sur la fréquence.

$$vitesse = \frac{4^{BRGH} \times Fosc}{64 \times (SPBRG + 1)} baud$$

Vitesse Kbaud	Fosc = 20 MHz			Fosc = 10 MHz			Fosc = 4 MHz			Fosc = 3.6864 MHz		
	kbaud	%err	SPBRG	kbaud	%err	SPBRG	kbaud	%err	SPBRG	kbaud	%err	SPBRG
300	-	-	-	-	-	-	0.300	0	207	0.3	0	191
1200	1.221	1.75	255	1.202	0.17	129	1.202	0.17	51	1.2	0	47
2400	2.404	0.17	129	2.404	0.17	64	2.404	0.17	25	2.4	0	23
9600	9.766	1.73	31	9.766	1.73	15	8.929	6.99	6	9.6	0	5
19200	19.531	1.72	15	19.531	1.72	7	20.833	8.51	2	19.2	0	2

Tableau IV.2 : vitesse de transmission ***BRGH = 0***

Vitesse Kbaud	Fosc = 20 MHz			Fosc = 10 MHz			Fosc = 4 MHz			Fosc = 3.6864 MHz		
	kbaud	%err	SPBRG	kbaud	%err	SPBRG	kbaud	%err	SPBRG	kbaud	%err	SPBRG
9600	9615	0.16	129	9615	0.16	64	9615	0.16	25	9600	0	23
19200	19231	0.16	64	19531	1.72	31	19231	0.16	12	19200	0	11
28800	29070	0.94	42	28409	1.36	21	27798	3.55	8	28800	0	7
33600	33784	0.55	36	32895	2.10	18	35714	6.29	6	32900	2.04	6

Tableau IV.3 : vitesse de transmission ***BRGH = 1***

V Le module MSSP (Master Synchronous Serial Port)

Le MSSP est une des deux modules de communication série du PIC 16F876/877. Il permet d'échanger des données en mode synchrone avec d'autres circuits qui peuvent être des microcontrôleurs, des mémoires EEPROM série, des convertisseurs A/N, des modules d'affichage . . . Il peut fonctionner selon deux modes : le mode SPI (Serial Peripheral Interface) et le mode I2C (Inter-Integrated Circuit)

V.1 Le mode SPI

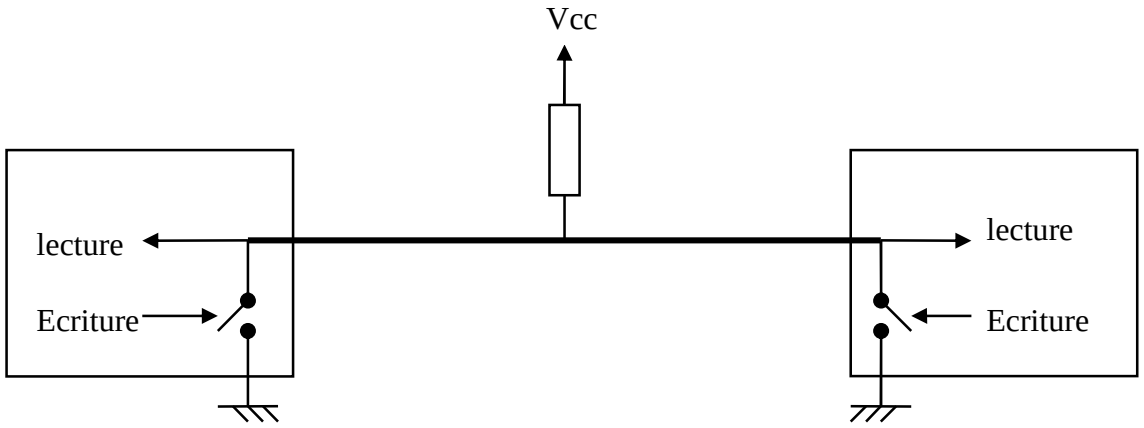
Le mode SPI (Serial Peripheral Interface) est étudié en Annexe II

V.2 Le mode I2C

Avant de parler du module SSP en mode I2C du PIC, introduisons très brièvement Le standard I2C.

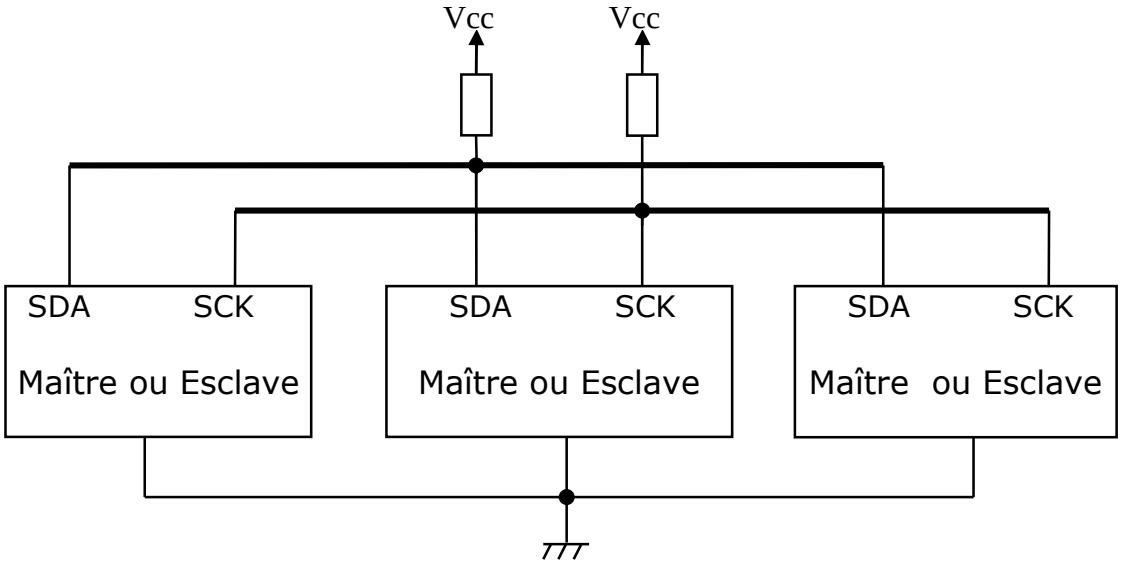
V.2.1 Introduction au bus I2C

Le bus I²C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement 3 fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse).
Comme les lignes SDA et SCK sont utilisées dans les deux sens par les deux circuits qui communiquent, on peut avoir un circuit qui place la ligne à 1 (Vcc) et l'autre qui la place à 0 (masse) ce qui correspond à un court circuit qui peut détruire les deux composants. Pour éviter ce problème, les E/S SDA et SCK fonctionnent en mode collecteur ouvert (ou drain ouvert) de sorte qu'un circuit ne peut imposer que le niveau bas ou ouvrir la ligne, le niveau haut est obtenu par une résistance de tirage externe. Ainsi une ligne est à 0 quand un des deux circuits impose le 0. Elle passe à 1 quand les deux circuits imposent le 1 (circuit ouvert). Le protocole I2C jongle avec cette situation pour organiser l'échange des données entre les deux composants.



Un bus I2C peut être relié à plusieurs circuits, mais pendant une communication, un des circuits est le maître, c'est lui génère l'horloge et initie les séquences de transmission, l'autre est l'esclave, il subit l'horloge du maître sur la ligne SCK mais il peut tout de même recevoir et émettre des données sur la ligne SDA. Chaque esclave a une adresse, au début d'une séquence de communication, le maître qui initie la séquence envoie l'adresse du slave avec

lequel il désire communiquer, celui-ci reconnaît son adresse et répond, les autres slaves (s'il y en a) restent muets.

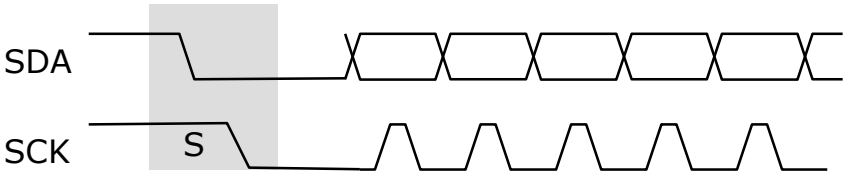


Certains circuits sont fabriqués pour être des masters, d'autres des slaves et d'autres peuvent être soit l'un soit l'autre.

Pour prendre le contrôle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1'). Lorsqu'un circuit prend le contrôle du bus, il en devient le maître. C'est lui qui génère le signal d'horloge et c'est lui qui initie les séquences d'échange.

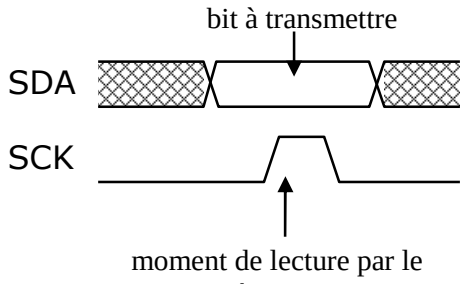
V.2.2 start condition

Au début d'une séquence d'échange, le master génère un start condition (S) avant de commencer l'échange de données. Au repos, les lignes SCL et SDA sont à l'état haut (relâchées). Pour générer un start, le master place d'abord la ligne SDA à 0, ensuite il place SCK à 0.



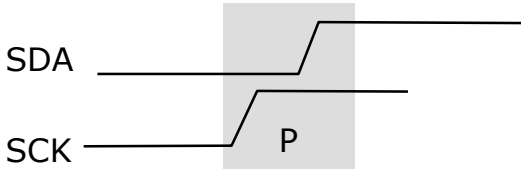
V.2.3 Transmission d'un bit

On place le bit à transmettre sur la ligne SDA ensuite on envoie une impulsion d'horloge sur la ligne SCK. C'est cette impulsion qui informe le slave qu'il doit lire la donnée sur SDA



V.2.4 Stop condition

A la fin d'une séquence d'échange, le master génère un stop condition (P) après lequel le bus est de nouveau libre. Pour cela, à partir de la situation SDA=0, SCK=0, le master commence par placer SCK à 1 et place ensuite SDA à 1.

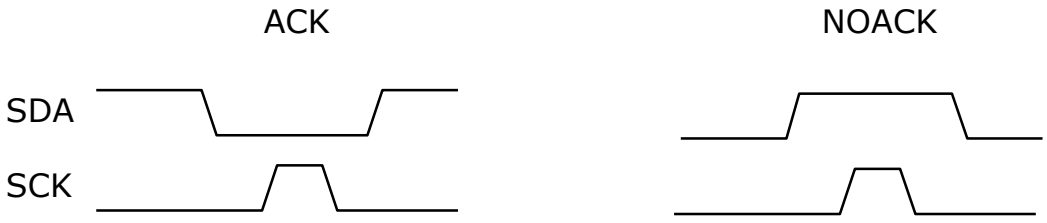


V.2.5 Remarque sur le Start et le Stop condition

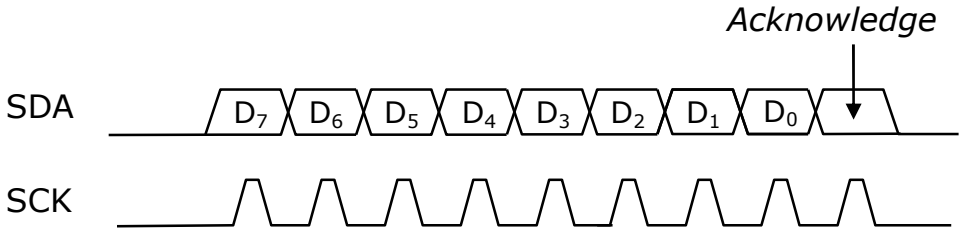
Une séquence de transmission peut contenir plusieurs *Starts conditions* avant de rencontrer un *Stop Condition*. On parle de *repeated Start condition*. Un Stop condition est toujours synonyme de FIN de transmission.

V.2.6 L' acknowledge

L' *acknowledge* est l'accusé de réception. Il est placé par le circuit qui reçoit sur la ligne SDA juste après la réception du 8ème bit. C'est l'émetteur qui le lit de la même façon qu'on lit un bit ordinaire. SDA=0 → *acknowledge* positif (ACK), SDA=1 → *acknowledge* négatif (NOACK),

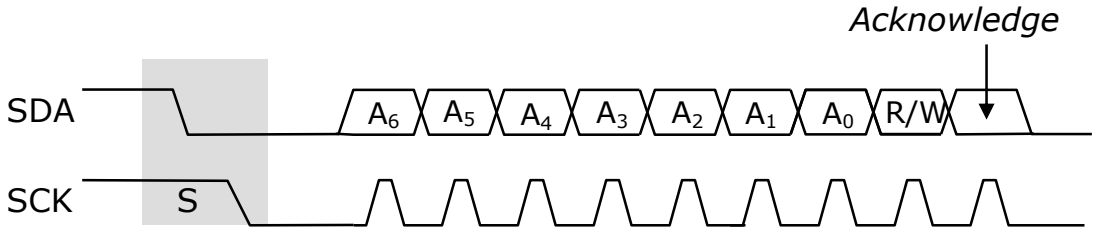


En conclusion, l'échange d'un octet nécessite 8 + 1 = 9 impulsions d'horloge sur la pin SCL.



V.2.7 L'adresse et le bit R/W

Comme on peut brancher plusieurs composant sur un bus I2C, il est nécessaire de définir une adresse unique pour chacun. Elle est codée sur 7 bits A_6 A_5 A_4 A_3 A_2 A_1 A_0. Le master qui démarre une séquence d'échange envoie l'adresse du slave juste après le *start condition*. Comme il y a seulement 7 bits, le master envoie à la 8ème position le bit R/W pour indiquer au slave s'il désire une émission (R/W=0) ou une réception (R/W=1).



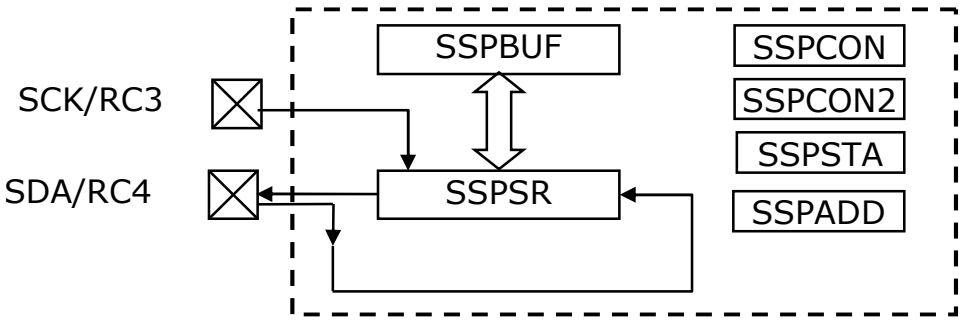
Le standard I2C supporte maintenant l'adressage sur 10 bits. Comme avec 7 bits on peut adresser jusqu'à 128 composant, cela nous suffira largement.

V.3 Le module MSSP en mode I2C

Le module MSSP du PIC peut être configuré en master ou en slave. Il utilise les broches RC3/SCL (Horloge) et RC4/SDA (données). Ces broches doivent être configurées en ENTREE à l'aide du registre TRISC et doivent être munie de résistances de pull-up externes nécessaire au fonctionnement I2C.

Les fréquences d'horloges supportées sont 100 kHz, 400 kHz et 1 MHz

L'accès au module en lecture et écriture se fait à l'aide du registre tampon (buffer) SSPBUFF. La transmission et la réception se fait à l'aide du registre à décalage SSPSR auquel nous n'avons pas directement accès



V.3.1 Transmission d'un octet

Pour transmettre un octet, il suffit de le copier dans le registre SSPBUF, et le module MSSP s'occupe du reste. Au moment de l'écriture dans SSPBUF, le bit BF passe à 1 et la transmission commence. A la fin de la transmission, le bit SSPSTAT.BF repasse à 0 et le drapeau d'interruption PIR1.SSPIF passe à 1.

Le bit BF apparaît donc comme un bit très important, c'est lui qui nous permet de savoir si le registre SSPBUF est libre ou non

Si on tante d'écrire dans SSPBUF alors que BF=1, le bit SSPCON.WCOL passe à 1 pour indiquer une collision et l'écriture n'a pas lieu.

V.3.2 Réception d'un octet

A la fin de la réception d'un octet, celui-ci est transféré dans SSPBUF, l'indicateur SSPSTAT.BF et le drapeau d'interruption PIR1.SSPIF passent à 1. BF repasse automatiquement à 0 au moment de la lecture de SSPBUF alors que SSPIF soit être remis à 0 par soft.

Si le PIC termine la réception d'un octet avant que l'octet précédent qui se trouve dans SSPBUF n'ait été lu, on a un Overflow qui sera signalé par le drapeau SSPOV. Le transfert n'a pas lieu, l'octet arrivé est perdu.

V.3.3 Les registres de configuration

Le control du module se fait à l'aide des registres :

SSPCON : registre de control
SSPCON2 : registre de control
SSPSTAT : registre d'état
SSPADD : registre d'adresse

SSPSTAT	SMP	CKE	D/A	P	S	R_W	UA	BF

SMP : Control de slew rate : 1 pour 100 kHz et 1 MHz, 0 pour 400 kHz

CKE : Control des niveau de tension à l'entrée, 0 : I2C, 1 : SMBUS

D/A : Indicateur d'état, 0 : dernier octet émis/reçu = adresse, 1 : dernier octet émis/reçu = donnée

P : Indicateur de Stop Condition. Ce bit passe à 1 quand on reçoit un Stop Condition. Il est automatiquement remis à 0 quand un autre événement et reçu.

S : Indicateur de Start Condition. Ce bit passe à 1 quand on reçoit un Start Condition. Il est automatiquement remis à 0 quand un autre événement et reçu

R_W : Indicateur de lecture écriture

Mode master : 0 : pas de transmission en cours, 1 : transmission en cours

Mode slave : image du bit R/W qui constitue le bit 0 du premier octet suivant le start condition. 0 : écriture, 1 : lecture

UA : Indicateur utilisé en mode esclave avec adressage 10 bits pour nous informer qu'il faut placer la 2ème partie de l'adresse dans le registre adresse. Ce dernier ayant seulement 8 bits, l'adresse est traitée en 2 parties. Ce bit est remis automatiquement à zéro après l'écriture dans SSPADD.

BF : Indicateur sur l'état du registre SSPBUF.

SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

WCOL : Collision d'écriture. En mode I2C master, ce bit passe à 1 si on tente d'écrire dans SSPBUF à mauvais moment de la séquence de transmission (BF=1). En mode I2C slave, il est positionné si on tente

d'écrire dans SSBUF avant la fin de la transmission de l'octet précédant. Ce bit doit être remis à zéro par programme.

SSPOV : Overflow, cet indicateur d'erreur est positionné quand une nouvelle donnée est reçue alors que le registre SSBUF n'est pas encore lu. Dans ce cas la nouvelle donnée est perdue. Ce bit doit être remis à zéro par programme.

SSPEN : Validation du module MSSP (1 = validé)

SKP : Utilisé en mode I2C slave pour générer des pauses en inhibant l'horloge.
0 : horloge forcée à zéro, 1 : horloge validée

SSPM3:SSPM0 : mode de fonctionnement du module
0110 : I2C Slave, adresse 7 bits
0111 : I2C Slave, adresse 10 bits
1000 : I2C master, fréquence = F_{osc} / (4 * (SSPADD+1))
1011 : I2C Slave forcé à l'état de repos
1110 : I2C master, adresse 7 bits, interruption sur START et STOP
1110 : I2C master, adresse 10 bits, interruption sur START et STOP

Le registre SSPCON2 :

SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN

Le bit 7 concerne le mode slave, les autres bits concernent le mode master :

GCEN : General call enable bit : Si ce bit est validé, le PIC répond à son adresse et à l'adresse générale 0000h

ACKSTAT : Acknowledge Status bit : En émission, ce bit est l'image de l'accusé de réception envoyé par le slave. 0=accusé positif, 1=accusé négatif. En réception, il permet au master de savoir si le slave a envoyé ou non un accusé de réception. 0 : ACK reçu, 1 : ACK non reçu

ACKDT : Acknowledge Data bit : Valeur du bit envoyé au slave en fin de réception. Ce bit est envoyé au moment ou on positionne le bit ACKEN.

ACKEN : Acknowledge Sequence Enable bit : quand il est placé à un, ce bit démarre une séquence accusé de réception qui consiste à placer les broches SDA et SCL dans le mode adéquat et à transmettre le bit ACKDT

RCEN : Receive Enable bit , 0 : réception inhibée, 1 : démarre la réception d'un octet

PEN : STOP Condition Enable bit , démarre une séquence stoP_condition sur les broches SDA et SCL. Ce bit est remis à 0 automatiquement

RSEN : Repeated START Condition Enable bit : démarre une séquence Repeated Start condition

SEN : START Condition Enable bit : démarre une séquence Start condition

Le registre SSPADD :

En mode I2C master, ce registre permet de déterminer la fréquence de communication

$$F = \frac{Fosc}{4 \times (SSPADD + 1)} \; Hz$$

Les fréquences possibles sont : 100 kHz, 400 kHz et 1 MHz

$$SSPADD = \frac{Fosc}{4 \times F} - 1 \; Hz$$

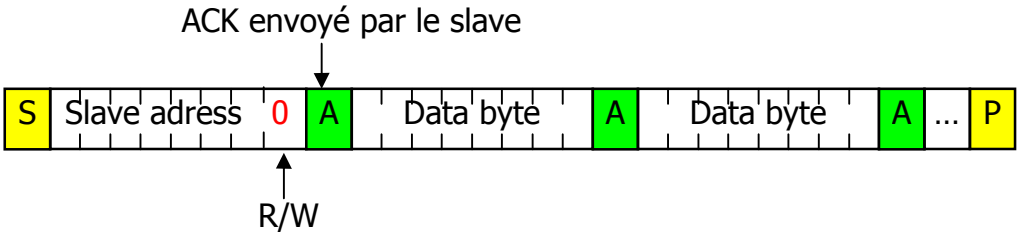
En mode I2C slave, ce registre doit contenir l'adresse du slave. Le bit 0 est réservé, il doit être toujours placé à zéro, l'adresse doit être écrite à partir du bit 1. En cas d'adresse 7 bits, cela ne pose pas de problème. En cas d'adresse 10 bits, l'adresse est écrite en 2 temps, On commence par écrire 1 1 1 1 0 A_9 A_8 0 dans SSPADD, en attend l'indicateur UA, ensuite on écrit A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

V.4 MSCP en mode Master

Une séquence de communication est toujours initiée par le master qui commence par envoyer un **S**tart_bit (SSPCON2.**S**EN), suivi de l'adresse du slave. L'adresse sera codée seulement sur 7 bits, Le LSB qu'on appelle bit R/W permet de préciser le sens de l'échange qui va suivre, R/W=0 pour transmission, R/W=1 pour réception. On envoie ou on reçoit les données et on termine la communication par un sto**P**_bit (SSPCON2.**P**EN)

Si l'on désire changer le sens de l'échange avant l'envoi du stoP, cela est possible, il faut envoyer un RepeatStart condition suivi de l'adresse accompagnée d'un R/W différent

V.4.1 Transmission master → slave :



- On positionne le bit SSPCON2.SEN sur le master, celui-ci envoie le Start bit sur la ligne SDA et r.à.z automatiquement le bit SEN. Le slave voit le Start_bit et se prépare à recevoir l'adresse,

- On copie l'adresse avec R/W=0 dans SSPBUF, les bits SSPSTAT.BF et SSPSTAT.R_W passent à 1, la transmission démarre et après le 8ème coup d'horloge, SSPSTAT.BF repasse à zéro, après le 9ème R_W repasse à 0, le drapeau d'interruption passe à 1 et l'horloge est forcée à 0 (pause). Le drapeau SSPIF doit être raz par soft pour qu'il puisse servir dans la suite.

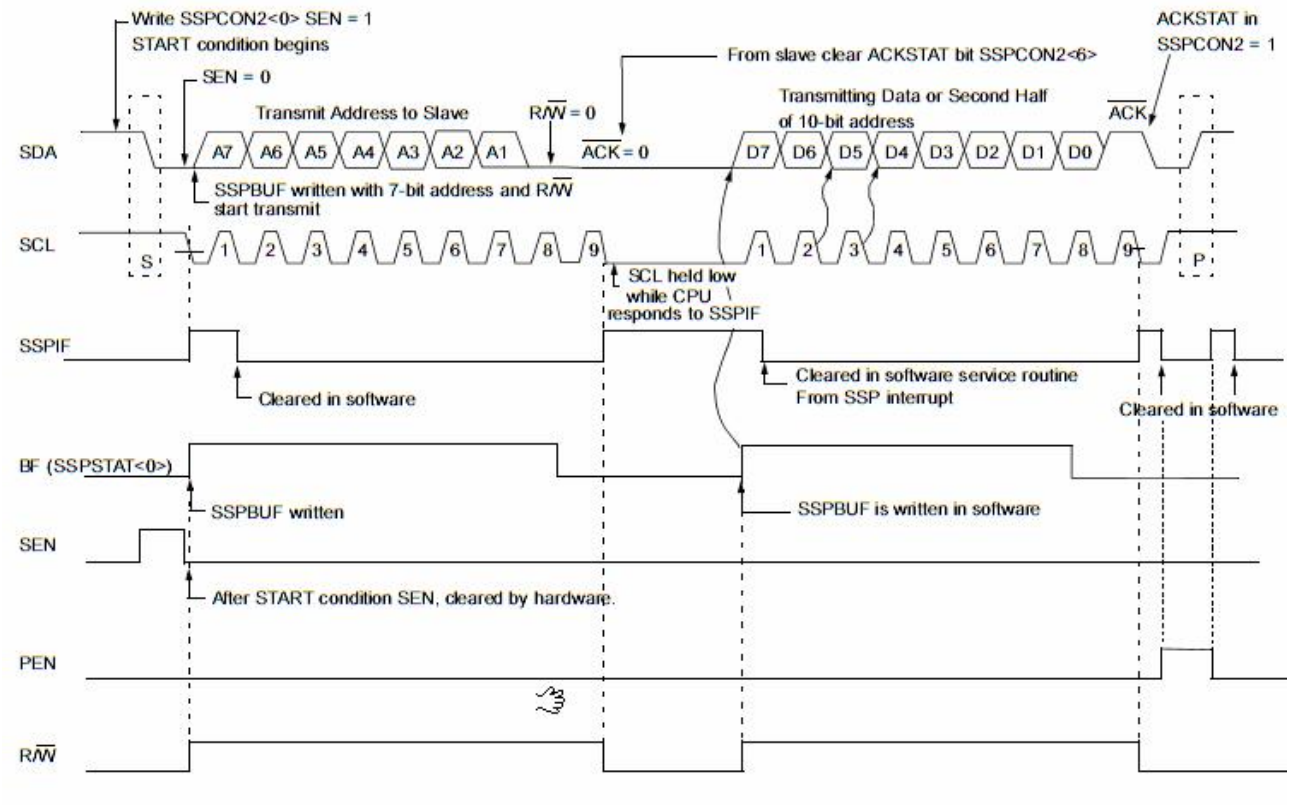
- Le slave détecte la fin de l'échange et envoie un Acquittement sur la ligne SDA : 0 (ACK) s'il a lu l'octet correctement, 1 (NoACK) s'il n'a pas pu lire l'octet correctement

- l'ACK renvoyé par le slave est recopié (par le master) dans le bit SSPCON2.ACKSTAT. Le programmeur doit vérifier l'état de ce bit pour décider de la suite des événements, Si = 0 (ACK), le programme peut envoyer une donnée vers le slave, sinon (NOACK) il faut envoyer un stoP_bit et recommencer,

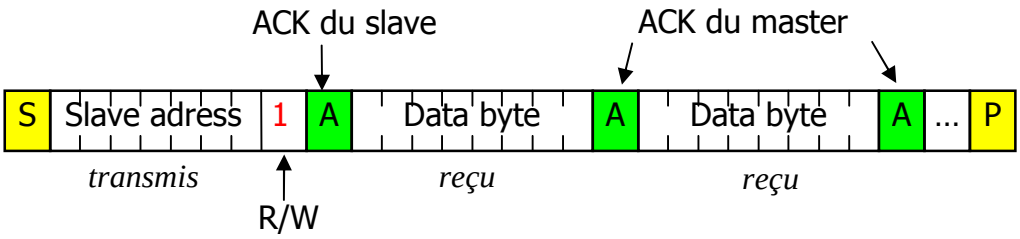
- Si SSPCON2.ACKSTAT=0, on envoie un octet de donnée en l'écrivant dans SSPBUF, Les bit BF et R_W passent à 1 au début de l'émission et repassent à 0 à la fin. Le drapeau SSPIF passe à 1. Le slave renvoie l'ACK/NOACK selon sa situation.

- Cet ACK/NOACK est accessible dans ACKSTAT, Si =ACK et on a encore des données à transmettre, on démarre une nouvelle émission en écrivant dans SSPBUF. Si =NOACK ou on n'a plus de données à transmettre, on envoie un stoP_bit en positionnant le bit SSPCON2.PEN qui est remis à zéro automatiquement après la transmission d'un Stop bit sur la ligne SDA

- Quand le slave voie le stoP_bit, il réinitialise son électronique pour se préparer à la réception d'un nouveau Start_bit.



V.4.2 Réception master ← slave :



- On positionne le bit SSPCON2.SEN sur le master, celui-ci envoie le Start_bit sur la ligne SDA et r.à.z automatiquement le bit SEN. Le slave voit le Start_bit et se prépare à recevoir l'adresse,

- On copie l'adresse avec R/W=1 dans SSPBUF, les bits SSPSTAT.BF et SSPSTAT.R_W passent à 1, la transmission démarre et après le 8ème coup d'horloge, BF repasse à zéro, après le 9ème R_W repasse à zéro et l'horloge est forcée à 0 (pause)

- A la fin du 8ème coup d'horloge, le slave renvoie un ACK/NOACK selon sa situation et force l'horloge à zéro (pause) pour empêcher le master de lire avant que la donnée ne soit préparée sur le bus

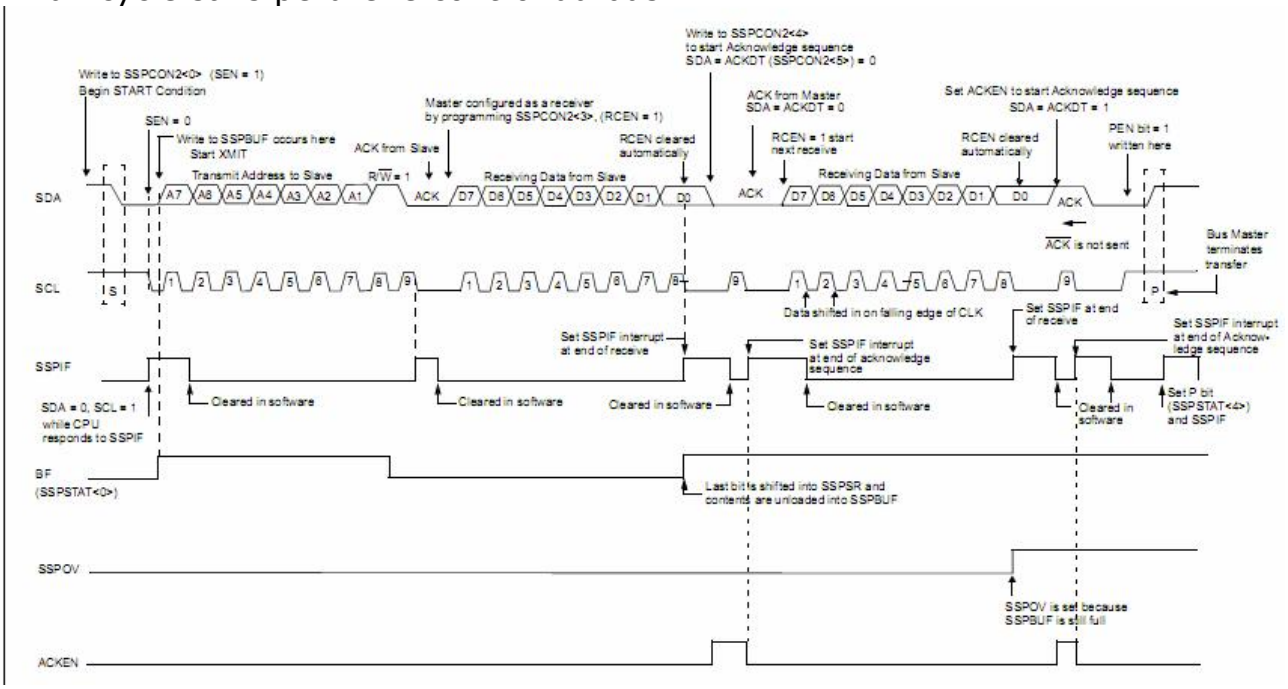
- Le programme du master doit tester l'acknowledge dans SSPCON2.ACKSTAT, si = 0, il se prépare à la réception en positionnant le bit SSPCON2.RCEN,

- Le slave prépare la donnée sur le bus et libère l'horloge, le Master le détecte, démarre l'horloge et la réception démarre
- A la fin de la réception, le bit SSPCON2.RCEN passe à 0, l'octet reçu est transféré dans SSPBUF, l'indicateur BF passe à 1 ainsi que le drapeau d'interruption PIR1.SSPIF. L'horloge est forcée à 0 (pause) pour donner le temps au programme utilisateur d'envoyer l' *Acknowledge,*

- Le programme utilisateur du master envoie l' *Acknowledge* en plaçant ACK/NOACK (0/1) dans la bit SSPCON2.ACKDT et en validant le bit SSPCON2.ACKEN. Le bit ACKDT est placé sur la ligne SDA, et ACKEN est r.a.z. automatiquement. L'horloge est libérée juste le temps d'un coup d'horloge (le 9ème) pour que le slave puisse prendre l' *Acknowledge* en compte. Elle est remise en pause après. Le drapeau d'interruption SSPIF est aussi positionné après l'envoi du ACK

- Si on a envoyé ACK, ça veut dire qu'on veut recevoir d'autres données, alors on initie la réception d'un nouvel octet en validant SSPCON2.RCEN.

- Si on a envoyé NOACK, (on ne veut plus de donnée), on doit envoyer un StoP_Cond pour terminer le cycle ou un Repeat Start_Cond pour démarrer un cycle sans perdre le control du bus.



V.4.3 Quelques remarques

- Une séquence d'échange démarre toujours par Start_condition suivi de l'adresse du slave accompagnée du bit R/W qui fixe le sens de l'échange
- L'indicateur BF repasse à 0 au moment de la lecture de SSPBUF
- Le drapeau d'interruption SSPIF doit à chaque fois être remis à 0 par soft. (ceci bien sur, si on a décidé de s'en servir)
- Quand on envoie un octet on reçoit toujours un ACK/NOACK de la part du slave (qu'on le teste ou non, ça c'est une autre affaire)
- Quand on reçoit un octet, on doit envoyer un ACK/NOACK vers le slave
- La direction de l'échange est décidée au moment de l'envoi de l'adresse à l'aide du bit R/W. Si on désire changer le sens de l'échange sans envoyer un stoP_condition pour éviter une perte éventuelle du control du bus, on envoie un Repeat_Start_Condition suivi de l'adresse accompagnée du nouveau R/W

V.5 MSCP en mode Slave

Voici quelques points en vrac :

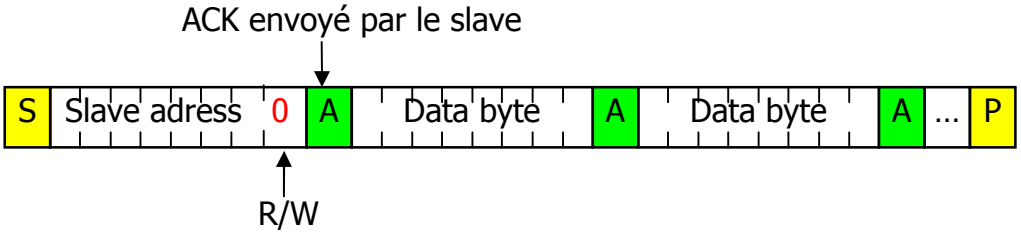
- Si le MSSP reçoit un Start_bit ou un RStart_bit, l'indicateur SSPSTAT.S passe à 1. Il reviendra à 0 automatiquement
- Si le MSSP reçoit un stoP_bit, l'indicateur SSPSTAT.P passe à 1. Il reviendra à 0 automatiquement
- Si le MSSP reçoit un ACK : il semble qu'il ne se passe rien
- Si le MSSP reçoit un NoACK : il réinitialise son électronique et se met en attente d'un nouveau Start_bit
- Si le MSSP reçoit un octet adresse (= à sa propre adresse)
    - l'indicateur SSPSTAT.D_A passe à 0
    - Le LSB (R/W) est copié dans l'indicateur R_W
    - Si SSPSTAT.BF=0 et SSPCON.SSPOV=0, l'octet est copié dans SSPBUF, le drapeau PIR1.SSPIF passe à 1, l'indicateur SSPSTAT.BF passe à 1 seulement dans le cas R/W=0
    - Si BF=1 et/ou SSPOV=1, SSPBUF n'est pas modifié, l'octet adresse est perdu, Les indicateurs BF, SSPOV et SSPIF passe à 1, Aucun ACK n'est transmis (NoACK).
- Si le MSSP reçoit un octet de donnée
    - l'indicateur SSPSTAT.D_A passe à 1
    - le tableau ci-dessous illustre les différentes possibilités. BF passe à un en cas de chargement de SSPBUF

Status Bits as Data Transfer is Received		SSPSR → SSPBUF	Generate $\overline{\text{ACK}}$ Pulse	Set bit SSPIF (SSP Interrupt occurs if enabled)
BF	SSPOV			
0	0	Yes	Yes	Yes
1	0	No	No	Yes
1	1	No	No	Yes
0	1	Yes	No	Yes

Note: Shaded cells show the conditions where the user software did not properly clear the overflow condition.

- Les ACK, comme on vient de le voir sont envoyés automatiquement
- Le drapeau BF repasse à 0 à la lecture du registre SSPBUF
- Le bit SSPOV doit être remis à 0 par instruction
- Le bit SKP permet de contrôler l'horloge
    - SKP=0, force l'horloge à l'état dominant 0 donc l'horloge est bloquée
    - SKP=1, force l'horloge à l'état récessif 1, donc l'horloge est libre

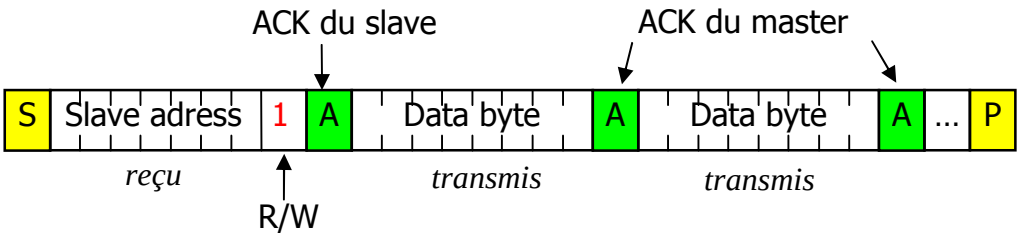
V.5.1 Réception de données (venant du master)



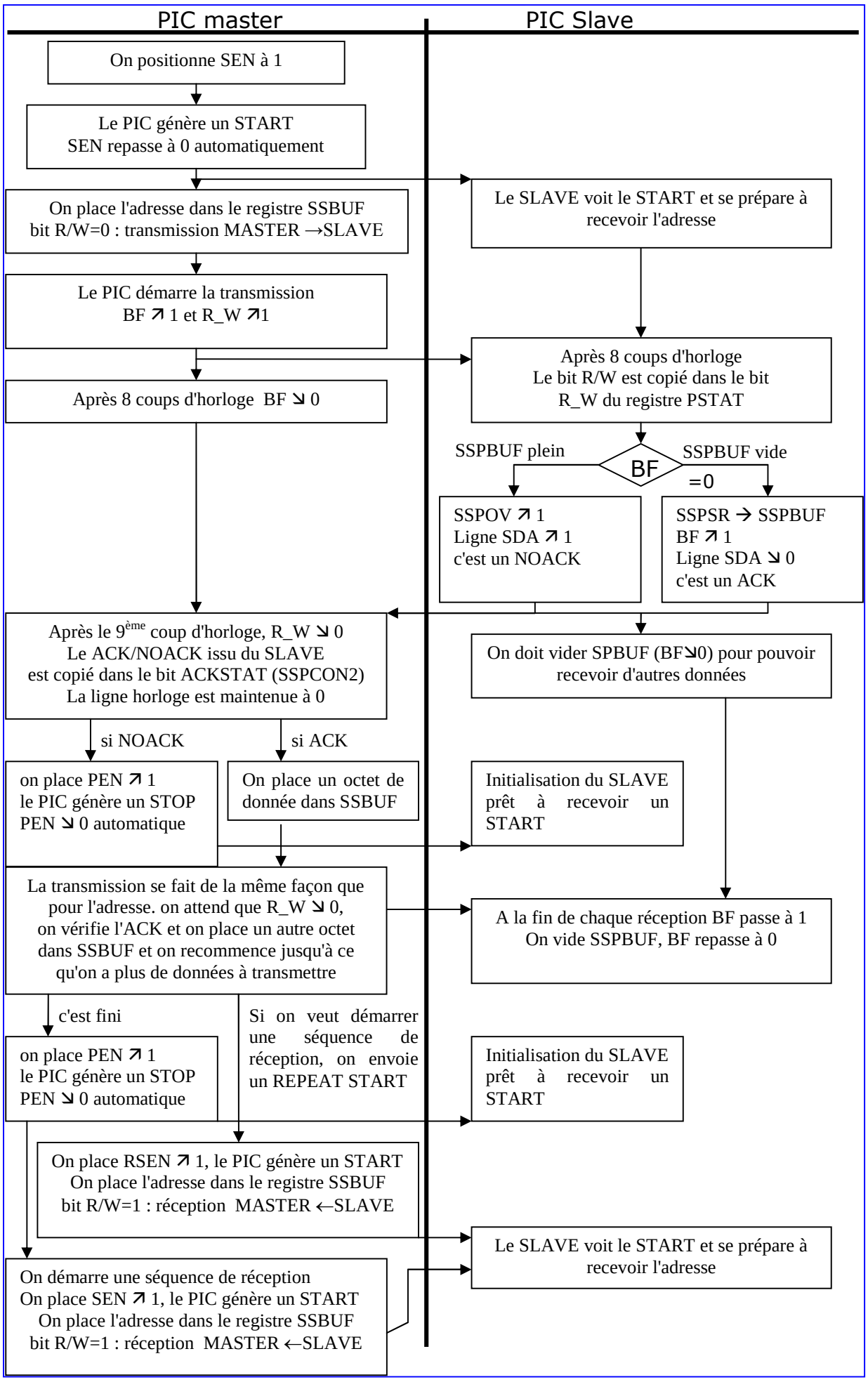
- Le PIC détecte le Start_bit sur la ligne SDA, positionne le bit SSPSTAT.S et se prépare à recevoir l'adresse,
- Le PIC détecte la fin de réception de l'adresse, il vérifie si elle correspond à son adresse.
- Si c'est l'adresse d'un autre slave, il ne réagit pas, il se met à attendre le stoP_condition,
- Si l'adresse reçue correspond à notre PIC. Il recopie le bit R/W (0 dans ce cas) dans le bit R_W du registre SSPSTAT , puis vérifie les bits BF et SSPOV pour savoir si le contenu précèdent du registre SSPBUF à été lu ou non
    - Supposons que tout est OK, BF=0 et SSPOV=0
        - Il recopie SSPSR (l'adresse) dans SSPBUF → BF passe à 1
        - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption
        - Le bit SSPSTAT.D_A passe à 0 ⇒ c'est une adresse qui est arrivée
        - Il place un ACK (0) sur la ligne SDA. C'est le PIC qui le fait, notre programme ne s'en occupe pas

- Notre programme détecte que BF ou SSPIF est passé à 1, il doit :
    - Vider SSPBUF pour que BF passe à 0 et qu'on soit prêt pour la suite
    - On vérifie le bit R_W, on trouve 0, on sait qu'on va recevoir un octet
- Le PIC attend la réception du 8ème bit, vérifie BF et SSPOV (on suppose que tout est OK), il transfert l'octet arrivé dans SSPBUF et place les bits D_A, BF et SSPIF à 1.
- Notre programme détecte BF ou SSPIF, il vide SSPBUF, BF repasse à 0
- Maintenant on peut soit recevoir, une autre donnée, soit un stoP_condition, soit un Repeat_Start_Condition. Il faut donc avoir l'œil, sur les indicateurs, BF, SSPIF, P et S pour décider de la suite des opérations

V.5.2 Transmission de données (vers le master)

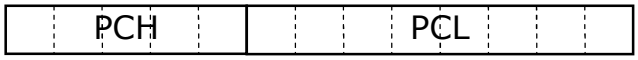


- Le PIC détecte le Start_bit sur la ligne SDA, positionne le bit SSPSTAT.S et se prépare à recevoir l'adresse,
- Le PIC détecte la fin de réception de l'adresse, il vérifie si elle correspond à son adresse.
- Si c'est l'adresse d'un autre slave, il ne réagit pas, il se met à attendre le stoP_condition,
- Si l'adresse reçue correspond à notre PIC. Il recopie le bit R/W (1 dans ce cas) dans le bit R_W du registre SSPSTAT , puis vérifie les bits BF et SSPOV pour savoir si le contenu précèdent du registre SSPBUF à été lu ou non
  - Supposons que tout est OK, BF=0 et SSPOV=0
    - Il recopie SSPSR (l'adresse) dans SSPBUF → **BF ne passe pas à 1**
    - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption
    - Le bit SSPSTAT.D_A passe à 0 ⇒ c'est une adresse qui est arrivée
    - Il place un ACK (0) sur la ligne SDA. C'est le PIC qui le fait, notre programme ne s'en occupe pas
- Notre programme détecte que SSPIF est passé à 1 :
  - On vérifie le bit R_W, on trouve 1, on sait qu'on va envoyer un octet
  - On n'a pas besoin de lire SSPBUF car BF est égal à 0
  - On place le bit CKP à 0 ce qui force l'horloge à 0 (pause) pour nous donner le temps de préparer la donnée.
  - On copie un octet de donnée dans SSPBUF. lindicateur SSPSTAT.BF passe à 1
  - On replace le bit CKP à 1, ce qui replace l'horloge à 1
  - Le master détecte, la libération de l'horloge, envoie une rafale de coup d'horloge, l'octet est transmis. L'indicateur SSPSTAT.BF repasse à 0 et le drapeau d'interruption PIR1.SSPIF passe à 1
  - Maintenant la master peut renvoyer soit un ACK, soit NoACK, suivi d'un stoP ou d'un RStart. Malheureusement on n'a aucun moyen (normal) au niveau de notre slave pour détecter l' ACK/NoACK, Bigonoff parle dans son cous d'une astuce pour le faire dans l'exemple qu'il a traité (cours : part2, rév 14, paragraphe 23.10). Il remarque que dans le cas ou le master renvoie un NoACK, l'indicateur R_W passe à 0. Si cela arrive on n'a plus qu'à attendre un Start_bit


PIC master
PIC Slave
On positionne SEN à 1
Le PIC génère un START
SEN repasse à 0 automatiquement
On place l'adresse dans le registre SSBUF
bit R/W=0 : transmission MASTER →SLAVE
Le PIC démarre la transmission
BF ↗ 1 et R_W ↗1
Après 8 coups d'horloge BF ↘ 0
Le SLAVE voit le START et se prépare à recevoir l'adresse
Après 8 coups d'horloge
Le bit R/W est copié dans le bit
R_W du registre PSTAT
SSPBUF plein
BF
SSPBUF vide
=0
SSPOV ↗ 1
Ligne SDA ↗ 1
c'est un NOACK
SSPSR → SSPBUF
BF ↗ 1
Ligne SDA ↘ 0
c'est un ACK
Après le 9ème coup d'horloge, R_W ↘ 0
Le ACK/NOACK issu du SLAVE
est copié dans le bit ACKSTAT (SSPCON2)
La ligne horloge est maintenue à 0
On doit vider SPBUF (BF↘0) pour pouvoir recevoir d'autres données
si NOACK
si ACK
on place PEN ↗ 1
le PIC génère un STOP
PEN ↘ 0 automatique
On place un octet de donnée dans SSBUF
Initialisation du SLAVE prêt à recevoir un START
La transmission se fait de la même façon que pour l'adresse. on attend que R_W ↘ 0, on vérifie l'ACK et on place un autre octet dans SSBUF et on recommence jusqu'à ce qu'on a plus de données à transmettre
A la fin de chaque réception BF passe à 1
On vide SSPBUF, BF repasse à 0
c'est fini
Si on veut démarrer une séquence de réception, on envoie un REPEAT START
on place PEN ↗ 1
le PIC génère un STOP
PEN ↘ 0 automatique
Initialisation du SLAVE prêt à recevoir un START
On place RSEN ↗ 1, le PIC génère un START
On place l'adresse dans le registre SSBUF
bit R/W=1 : réception MASTER ←SLAVE
Le SLAVE voit le START et se prépare à recevoir l'adresse
On démarre une séquence de réception
On place SEN ↗ 1, le PIC génère un START
On place l'adresse dans le registre SSBUF
bit R/W=1 : réception MASTER ←SLAVE

VI Annexe : Gestion du Programme Counter,

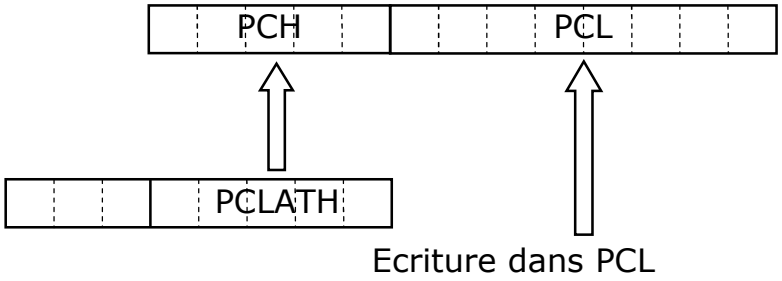
Le PC est un registre de 13 bits, il peut donc adresser 8 k de mémoire programme.



PCL (8 bits) est la partie basse de PC, il est accessible en lecture écriture
PCH (5 bits) est la partie haute de PC, il n'est pas accessible directement. On peut toutefois le modifier indirectement à l'aide du registre PCLATH qui est une registre SFR où seuls 5 bits sont utilisé.

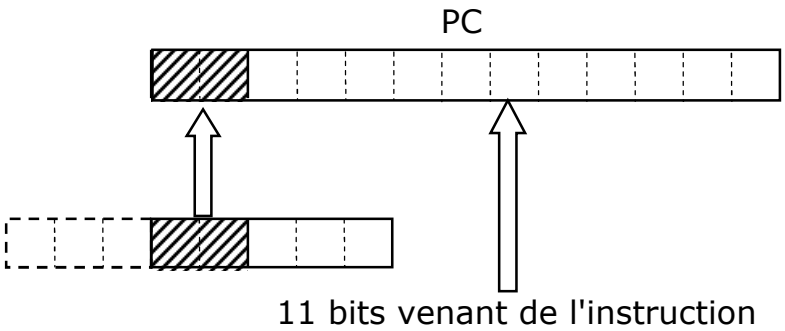
VI.1 GOTO calculé : modification de la valeur de PCL

Quand on écrit quelque chose dans PCL, alors le contenu de PCLATH est automatiquement copié dans PCH. Donc, avant d'écrire dans PCL s'assurer que la valeur contenue dans PCLATH est correcte.



VI.2 Instruction de branchement

Un branchement consiste à modifier la valeur de PC. Mais dans les instructions de branchement, seulement 11 bits sont utilisés pour coder l'adresse de destination, ce qui donne au maximum un intervalle de 2 k alors que nous avons une mémoire programme de 8k. Pour remédier à ce problème, les deux bits manquants pour compléter les 13 bits de PC sont pris dans PCLATH :



La mémoire programme apparaît donc comme organisée en 4 page de 2 k chacune. Lors d'un saut, l'adresse instruction précise la destination à l'intérieur d'une page et les deux bits 3 et 4 de PCLATH désignent la page

Attention, en assembleur on ne précise pas les adresses mais on utilise des étiquettes qui correspondent à des adresses complètes (13 bits). Cela ne

change rient car seuls 11 bits de l'étiquette seront envoyés dans le PC et c'est à nous de gérer les deux bits de PCLATH pour compléter l'adresse de destination.

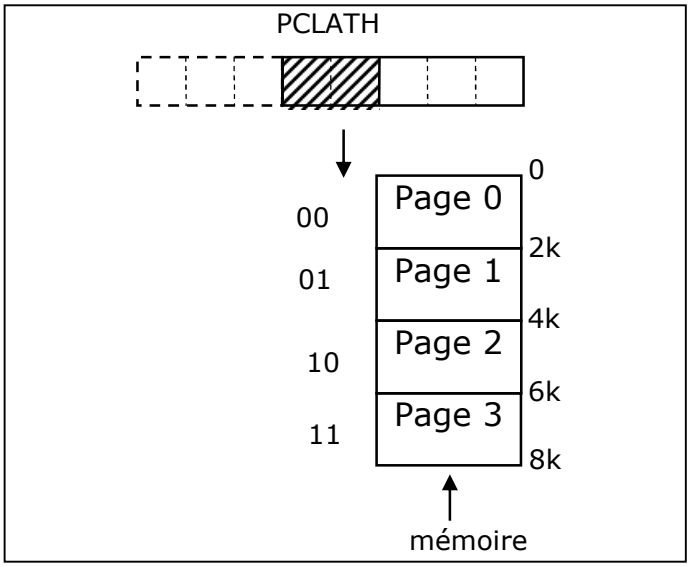


Fig. VI.1 : gestion de la mémoire programme lors d'une instruction de saut

VII Références

1) http://www.multimania.com/pmorphe/bienvenuechez_atm.htm
2) La programmation des PICs, Première partie – PIC16F84, Rev 5, Bigonff
3) La programmation des PICs, Secode partie-PIC16F876/877, Rev 7, Bigonff
4) PIC 16F87X Data Sheet (DS30292C) , Microchip
5) PICmicro™ Mid-Range MCU Family Reference Manual (DS33023A), Microchip
6) La page web de Jacques Weiss sur les PIC
7) La page web d'Offset sur les PIC