TABLE DES MATIERES

1. Introduction :

1.1. Présentation de la famille PIC :

Il existe 3 grandes catégories de microcôntroleurs PIC :

- la famille ***PIC16C5X*** : ***33 instructions***, instructions sur ***12 bits***, ***prix très bas***, pas d'interruptions, 18 ou 28 broches. Cette famille est la plus ancienne et la plus répandue. C'est grâce à cette famille et notamment le PIC16C54 que Microchip a pu se placer et dominer le marché des produits « grand public » (four à micro-onde, machine à laver, console de jeu, ...).
- la famille ***PIC16CXX*** : ***35 instructions***, instructions sur ***14 bits***, ***milieu de gamme***. Cette famille qui n'est malheureusement pas compatible avec la précédente est beaucoup plus complète. Par exemple les PIC16C7X possèdent un C.A.N. intégré multiplexé (4 ou 8 voies) et le PIC16C84 et le PIC16F84 ont leur mémoire programme respectivement en EEPROM et en FLASH (Cette technologie permet la programmation du composant IN SITU, c'est à dire sans manipulation).
- la famille ***PIC17CXX*** : ***35 instructions***, instructions sur ***16 bits***, mémoire programme extensible à 64 k*16bits. Ce microcôntroleur ressemble à des microcôntroleurs 8 bits classiques type 80C51, avec une différence essentielle : le nombre d'instructions (microcôntroleur à structure RISC).

Remarque : il existe aussi la famille ***PIC12CXX***, 33 instructions, instructions sur 12 bits et 8 broches seulement. Cette famille convient aux applications intelligentes ne demandant que très peu d'entrées/sorties.

Un extrait du catalogue Micochip est donné à la figure 1 :

		Maximum Frequency of Operation (MHz)	Program Memory: EPROM	Program Memory: ROM	Program Memory: EEPROM	Data Memory (bytes)	Data EEPROM (bytes)	Timer Module(s) †	Capture/Compare/PWM Module	Serial Port (s) (SPI/I²C, SCI)	Parallel Slave Port	Analog to Digital Converter (8-bit)	External Interrupts	Interrupts Sources	I/O	Voltage Range (Volts)	Number of Instructions	Packages
High-end	PIC17C42§	25	2K	—	—	128	—	TMR0, TMR1, TMR2,TMR3	—	SCI	—	—	Yes	11	33	4.5 - 5.5	55	40-pin DIP, 44-pin PLCC 44 pin QFP
Mid-range	PIC16C64	20	2K	—	—	128	—	TMR0, TMR0, TMR2	1	SPI/I²C	Yes	—	Yes	6	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C71	16	1K	—	—	36	—	TMR0	—	—	—	4 ch	Yes	4	13	3.0 - 6.0	35	18-pin DIP, 18-pin SOIC 44 pin QFP
	PIC16C74	20	4K	—	—	192	—	TMR0, TMR1, TMR2	2	SPI/I²C, SCI	Yes	8 ch	Yes	12	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C84	10	—	—	1K	36	64	TMR0	—	—	—	—	Yes	4	13	2.0 - 6.0	35	18 pin DIP, 18 pin SOIC 20-pin SSOP
Base Line	PIC16C54	20	512	—	—	25	—	RTCC	—	—	—	—	—	0	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C54A	20	512	—	—	25	—	RTCC	—	—	—	—	—	0	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16CR54	20	—	512	—	25	—	RTCC	—	—	—	—	—	0	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C55	20	1K	—	—	25	—	RTCC	—	—	—	—	—	0	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C56	20	1K	—	—	25	—	RTCC	—	—	—	—	—	0	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C57	20	2K	—	—	72	—	RTCC	—	—	—	—	—	0	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16CR57A	20	—	2K	—	72	—	RTCC	—	—	—	—	—	0	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C58A	20	2K	—	—	72	—	RTCC	—	—	—	—	—	0	12	2.5	33	18-pin DIP, 18-pin SOIC

fig.1 :famille PIC16/17.

1.2. Les caractéristiques de la famille PIC :

Les principaux avantages de la famille PIC sont les suivants :

- structure interne statique ce qui leur permet un arrêt de l'horloge (instruction SLEEP) pour une économie d'énergie dans les applications embarquées,
- jeu d'instructions réduit (33 ou 35 instructions),ce qui permet un apprentissage rapide,
- microcôntroleur rapide puisque 1 instruction est exécutée en 4 cycles d'horloge (sauf les sauts),
- keeloq system permet une confidentialité.

1.3. Les applications :

L'arrivée des microcôntroleurs à structure R.I.S.C. sur le marché, au début des années 90 et notamment l'arrivée des PIC a vu l'apparition de nouveaux marchés qui auparavant n'étaient pas investis par les microcôntroleurs. Ainsi une politique de prix agressive, alliée à un produit innovant, a permis aux PIC de se retrouver dans les téléviseurs, les fours à micro-onde et dans beaucoup de produits « grand public » qui auparavant ne comportaient pas de microcôntroleurs. Chez les industriels aussi, des cartes logiques qui ne comportaient pas de microcôntroleurs ont été envahis par les PIC et autres microcôntroleurs à structure R.I.S.C.

Les applications à base de PIC sont nombreuses :

- remplacement de circuits spécifiques (gain d'argent),
- remplacement de batterie de circuits logiques (gain de place),
- application de contrôle moteur,
- filtrage, transmission / réception,
- automatismes,
- applications portables,
- applications qui demandent une confidentialité du circuit,
- cartes spécifiques pour PC.

1.4. Les 3 différentes options de mémoire programme :

PIC1X***C***XX : la mémoire programme est en ***EPROM***. Pour ce type de microcôntroleur, il existe 2 solutions : le boitier comporte une ***fenêtre d'effacement*** (plus cher, mais permet le développement du programme avec des cycles d'effacement par UV et écriture) ou le boitier est dit « ***O.T.P*** » (One Time Programmable), moins cher mais ne permet pas la réécriture du programme.

PIC1X***CR***XX : la mémoire programme est en ***ROM*** et est donc programmée en usine par le fondeur. Ces circuits sont utilisés pour la grande série.

PIC16***F***XX : la mémoire programme est une mémoire ***FLASH*** ce qui permet un cycle d'effacement très rapide et sur toute la mémoire (flash).

1.5. Les outils de développement :

Les ***logiciels de développement*** sont les suivants :

MPASM : cross-assembleur ***universel*** pour PIC 16/17, cet assembleur permet l'utilisation de macros et fournit un fichier objet et un fichier listing.

MPLAC : cross-assembleur pour PIC 16CXX, cet assembleur permet l'utilisation macros et fournit un fichier objet et un fichier listing.

MPSIM : simulateur de PIC. Cet outil est très utile pour l'auto formation sur le produit.

MP-C : compilateur C pour la famille PIC 16/17.

Il existe 2 principaux types d'***outils de développement*** : les programmateurs et les émulateurs dont quelques-uns sont listés ci-dessous :

le ***PICSTART PLUS*** : programmateur de PIC16CXX (bon marché)
le ***PROMATE II*** : programmateur universel (PIC 16/17)
le ***PICMASTER*** : émulateur haut de gamme.

2. *Le PIC16C5X :*

2.1. *Présentation :*

Le brochage des 2 types de PIC 16C5X est donné à la figure 2.

On remarquera la broche MCLR (Master Clear ou broche de reset), la broche RTCC (entrée du compteur) et les ports d'entrée/sortie notés RAi, RBi et RCi

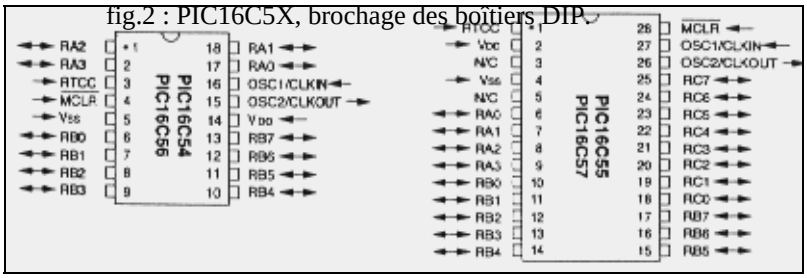


fig.2 : PIC16C5X, brochage des boîtiers DIP.

Les principales caractéristiques de la famille PIC16C5X sont résumés à la figure 3. La mémoire de données RAM comporte les registres à fonction spéciale (SFR) et la mémoire utilisateur.

PIC	*EPROM*	*RAM*	*Ports d'E/S*
PIC 16C54	512 x 12	32 x 8	12
PIC 16C55	512 x 12	32 x 8	20
PIC 16C56	1k x 12	32 x 8	12
PIC 16C57	2k x 12	80 x 8	20
PIC 16C58	2k x 12	80 x 8	12

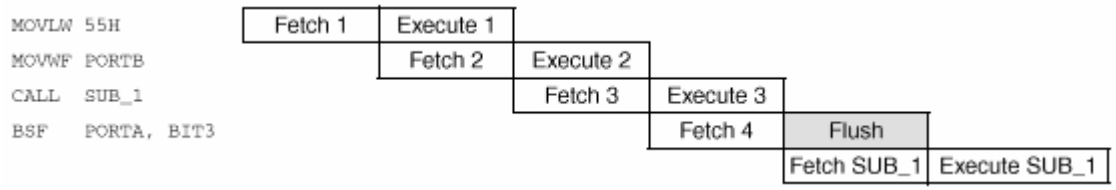
fig.3 : caractéristiques principales des membres de la famille PIC16C5X.

Les composants de la série se distinguent aussi par la cadence paramétrable de leur oscillateur. Le choix d'un des modes de fonctionnement se fait par 2 bits de configurations programmé par l'utilisateur. Les quatre possibilités sont données à la figure 4. Pour plus de détail, on se reportera à la notice constructeur.

Identification	*fréquence MIN*	*fréquence MAX*	*Commentaire*
LP	DC	32 kHz	Low Power
RC	DC	4 MHz	Oscillateur RC
XT	DC	4 MHz	Oscillateur à quartz
HS	DC	20 MHz	High Speed

fig.4 : cadence de fonctionnement.

Une instruction est exécutée en 4 cycles machine sauf pour les sauts. En effet, grâce à sa structure de Harvard (bus de données et bus d'adresse séparés), le microcôntroleur utilise un pipe-line à 2 couches. Pendant le premier cycle il va cherché l'instruction suivante (Fetch) et il exécute l'instruction courante (Execute). Cette chaîne est cassée lors de saut, et il faut donc faire un fetch et ensuite un execute, ce qui prend 8 cycles.



2.2. Mémoire programme :

La mémoire programme est organisée en pages de 512 mots d'instruction chacune. Les PIC16C54 et PIC16C55 ne disposent que d'une page, le PIC16C56 en a deux, sélectionnées par le bit 5 du registre d'état STATUS et le PIC16C57 en a quatre, sélectionnées par les bits 6-5 de STATUS. Les figures 5, 6 et 7 nous montre ce principe.

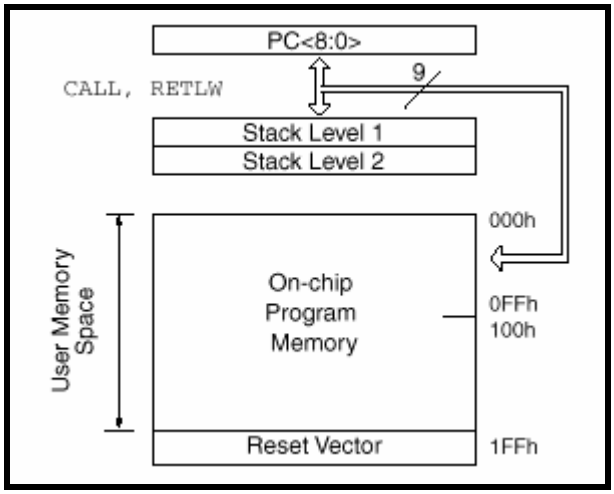


fig.5 :Mémoire programme du PIC16C54 et PIC16C55.

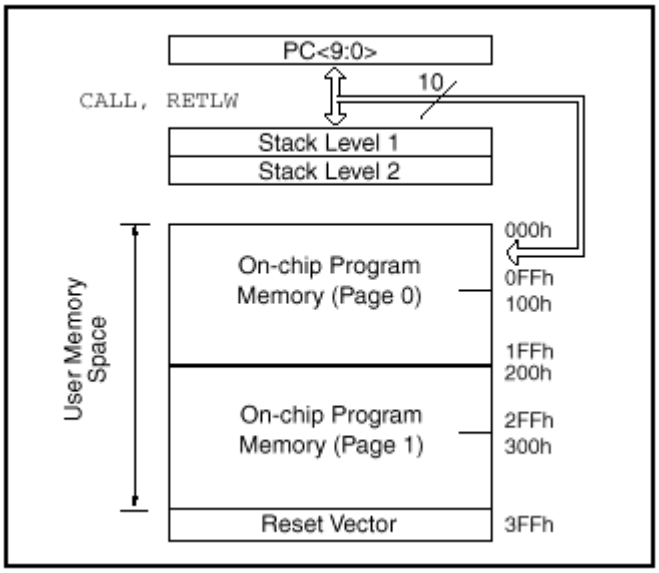


fig.6 :Mémoire programme du PIC16C56.

Les PIC16C54 et PIC16C55 ont un PC (Program Counter) de 9 bits ce qui permet un adressage de 512 x 12 bits.

Le PIC16C56 a un PC de 10 bits (adressage de 1k x 12 bits).

Les PIC16C57 et PIC16C58 ont un PC de 11 bits (adressage de 2k x 12 bits)

Au reset tous les bits du PC sont placés à 1.

Les PIC16C5X possèdent une pile matérielle de seulement 2 niveaux. Il ne pourra donc y avoir au maximum que 2 appels de procédure imbriqués.

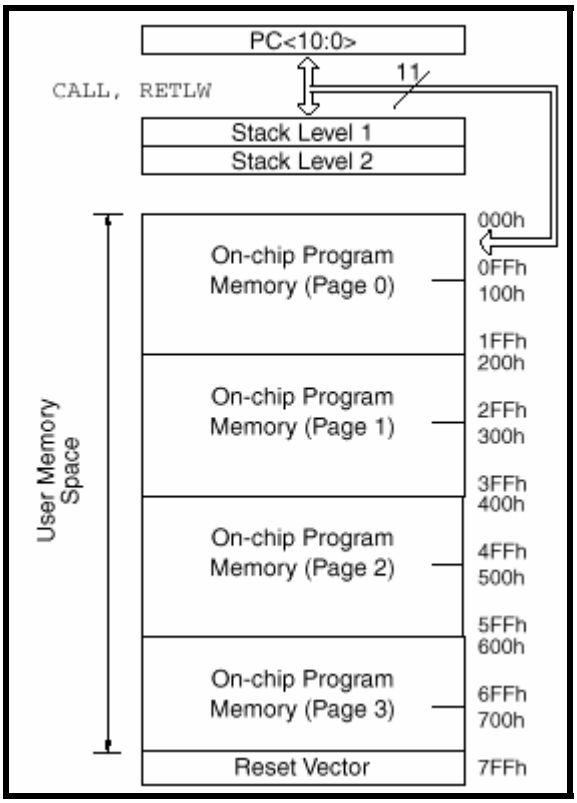


fig.7 :Mémoire programme du PIC16C57 et PIC16C58.

2.3. Mémoire de données :

La mémoire de données (RAM) est composée de 32 octets (PIC16C54, PIC16C55 et PIC16C56) ou de 80 octets (PIC16C57 et PIC16C58). Les figures 8 et 9 donnent l'image de la RAM.

La mémoire de données est divisée en 2 groupes fonctionnels :

- les registres du SFR (Special Function Register) et
- les registres du GPR (Group General Register) ou registres utilisateur.

Les registres du SFR sont composés de ***TMR0*** (registre Timer), de ***PCL*** (Pointeur de programme 8 bits), de ***STATUS*** (registre d'état), de ***FSR*** et ***INDF*** (adressage indirect) et des registres ***PORTA***, ***PORTB*** et/ou ***PORTC*** (ports d'entrées/sorties).

Les registres du GPR sont directement utilisables par le programmeur.

Chaque bit des registres de la RAM peut être forcé à 0, à 1 ou peut être testé pour un saut conditionnel.

Contrairement aux autres microcôntroleurs (type 8051) on ne fait pas la différence entre registres et mémoire de données. Les instructions sont les mêmes dans les deux cas.

Il existe 4 registres (***TRIS A TRIS B et TRIS C et OPTION***) qui ne font pas partie de l'espace adressable et qui seront écrit par, respectivement, l'instruction TRIS et OPTION. Les registres TRIS A, TRIS B et TRIS C permettent de configurer respectivement les ports A, B et C en haute impédance ou non (configuration en entrée ou en sortie des ports). Le registre OPTION paramètre le timer.

Le registre ***W*** (Working Register) plus connu sous le nom d'accumulateur ne fait pas parti des registres du SFR. Il est le passage obligé pour certaines instructions.

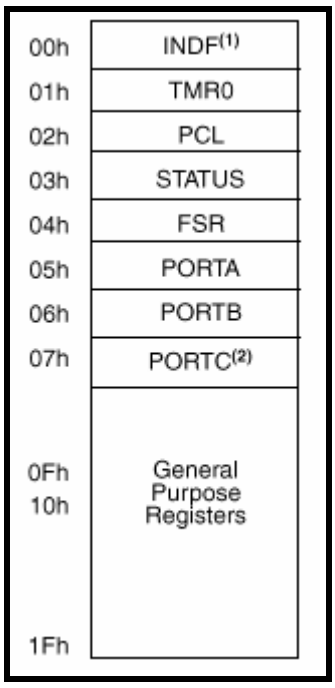


fig.8 : Mémoire de données du PIC16C54, PIC16C55 et PIC16C56.

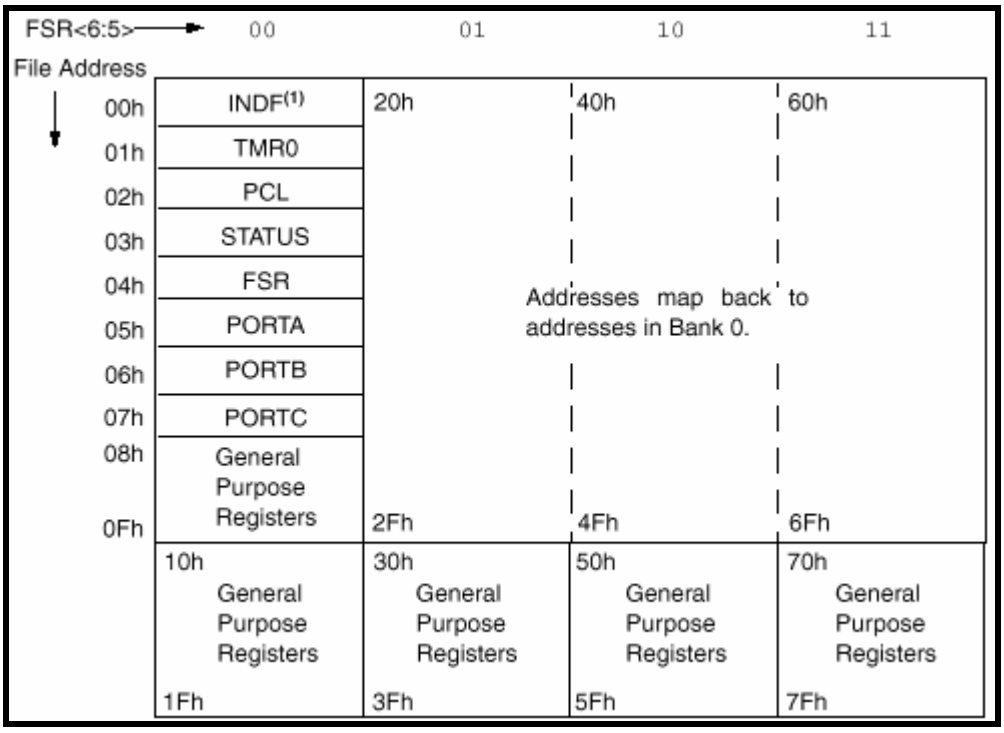


fig.9 : Mémoire de données du PIC16C57. Le PIC16C58 ne comporte pas de PORT C.

La mémoire de donnée pour les PIC16C57 et PIC16C58 est une mémoire paginée, car une instruction en adressage direct ne peut accéder qu'à 32 registres (opérande codée sur 5 bits, Cf chapitre 2.4). Il faudra donc, pour accéder aux registres d'adresse supérieure à 1Fh, utiliser les ***bits 6 et 5 du registre SFR***. On remarquera à la figure 9 que seuls les registres aux adresses 30h-3Fh, 50h-5Fh et 70h-7Fh sont utilisables par le programmeur. Les registres aux adresses 20h-2Fh, 40h-4Fh et 60h-6Fh sont « mappés » en mémoire 00h-1Fh et ceci pour accéder quel que soit la page sélectionnée aux registres du S.F.R.

2.4. Les modes d'adressage :

Il existe 3 modes d'adressage : adressage direct, adressage indirect et adressage immédiat.

2.4.1. Adressage direct :

Exemple : **MOVWF FSR ; équivalent à MOVWF 04H : W → FSR**

Rappel : Si le microcôntroleur est un PIC16C57 ou PIC16C58, la mémoire de donnée est une mémoire paginée, et ce sont les bits 5 et 6 du registre ***FSR*** qui choisissent la page, voir figure 10.

Remarque : quel que soit la page sélectionnée, les registres du SFR et GPR (adresses comprises entre 00h et 0Fh) restent inchangés. Seuls les registres du GPR compris entre l'adresse 10h et 1Fh sont paginés.

2.4.2. Adressage indirect :

Il n'y a pas d'instructions spéciales pour spécifier l'adressage indirect. Cela se fait grâce au registre INDF (adresse 00h). Le fait d'utiliser ce registre comme opérande permet au décodeur d'instruction de savoir que l'on fait de l'adressage indexé. C'est le registre ***FSR*** qui sera utilisé pour pointer vers l'adresse de la valeur à lire ou à écrire. On utilisera les bits <4..0> du SFR pour les PIC16C54, PIC16C55 et PIC16C56. Pour les PIC16C57 et PIC16C58 on utilisera les bits <6..0> pour atteindre tout l'espace adressable. (Cf figure 10)

Exemple 1 : **MOVWF INDF ; placer le contenu de W dans le registre pointé par FSR : W → [FSR]**

Exemple 2 : on veut initialiser toute la RAM à partir de l'adresse 10h jusqu'à l'adresse 1Fh.

```
        MOVLW   0X10    ; initialisation du pointeur
        MOVWF   FSR     ; vers le début de la RAM
NEXT    CLRF    INDF    ; mettre à 0 le registre pointé par FSR
        INC     FSR, F  ; octet suivant
        BTFSC   FSR, 4  ; fini ?
        GOTO    NEXT    ; non
FIN
```

2.4.3. Adressage immédiat :

Exemple : **MOVLW 25 ; Move litteral vers W : 25 → W**

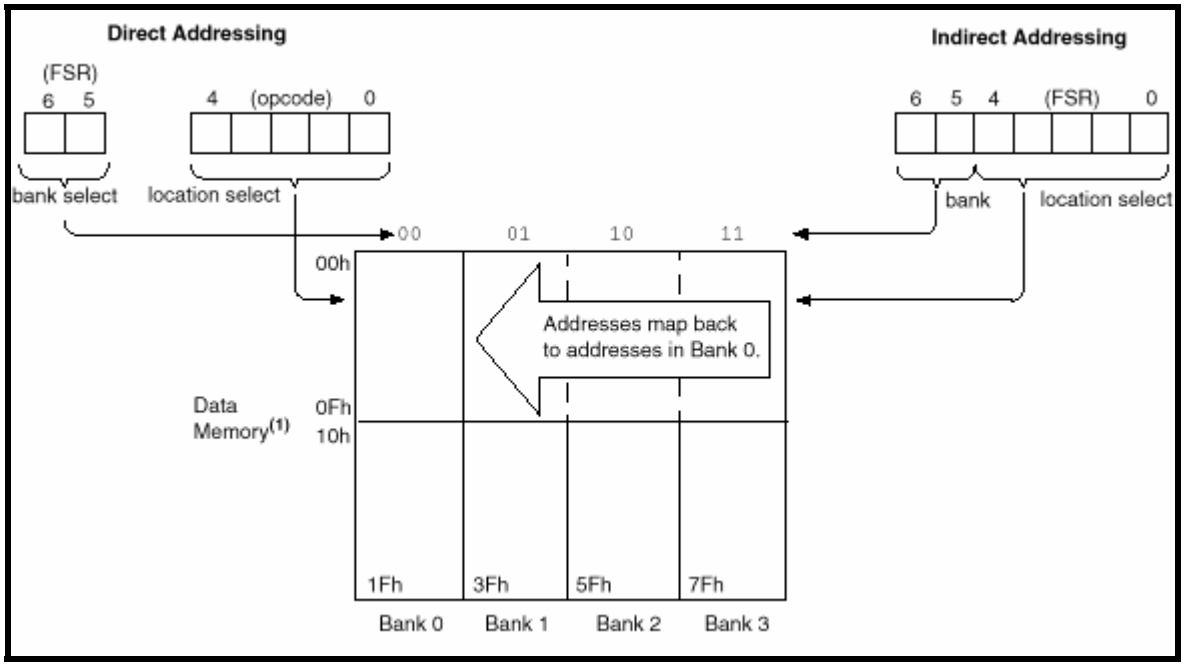
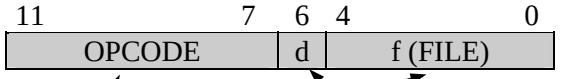


fig. 10 : adressage direct et indirect.

2.5. Codage des instructions :

Les instructions sont données en ANNEXE. On peut les regrouper en 4 groupes d'instruction :

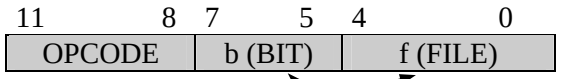
2.5.1. Instructions sur les octets :



d = 0 destination W
d = 1 destination F
f = registre dans l'espace adressable 32 octets

Exemple : **ADDWF PORTA, 0 ; W + PORT A → W**

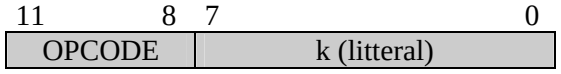
2.5.2. Instructions sur les bits :



b = 3 bits (adresse du bit)
f = registre dans l'espace adressable 32 octets

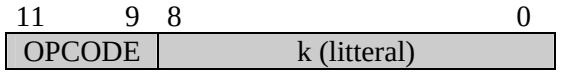
Exemple : **BSF 0x14, 5 ; mise à 1 du bit 5 du registre à l'adresse 14h**

2.5.3. Instructions adressage immédiat et CALL :



Exemples : **ANDLW B'00100010' ; W ← W ET 00100010**
CALL SUB ; SUB codé sur 8 bits

2.5.4. instruction GOTO :



Exemple : **GOTO SAUT ; SAUT codé sur 9 bits**

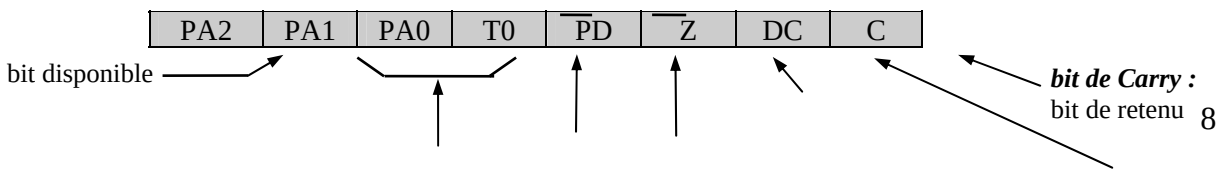
Comme on le verra dans le chapitre suivant, l'instruction GOTO permet d'accéder à un espace plus important (une page de 512 instructions) que l'appel à la procédure CALL (une demi-page de 256 instructions). Cette limitation n'existe plus dans les PIC de milieu de gamme (PIC16CXX) ou les instructions GOTO et CALL permettent d'accéder à un espace de 2k x instructions de 14 bits.

2.6. Les registres du SFR :

2.6.1. W : Working Register

L'accumulateur reçoit les valeurs littérales. Il est aussi utilisé pour écrire dans les registres TRIS et OPTION (seule façon d'y accéder). Le bit noté d pour les instructions sur les octets permet, de plus, de spécifier la direction du résultat (vers W ou vers le registre F). Pour plus de détail, voir en ANNEXE (Architecture matérielle du PIC 16C5X).

2.6.2. STATUS : registre d'état



Bits de sélection de page :
PIC 16C54/C55 :PA1 PA0 disponibles
PIC 16C56 : PA1 bit disponible
PA0 0 = page 0 (000_H-$1FF_H$)
1 = page 1 (200_H-$3FF_H$)
PIC 16C57/C58 : PA1 PA0
0 0 = page 0 (000_H-$1FF_H$)
0 1 = page 1 (200_H-$3FF_H$)
1 0 = page 2 (400_H-$5FF_H$)
1 1 = page 3 (600_H-$7FF_H$)

bit de Time Out :
mis à 1 après une mise sous tension et par les instructions CLRWDT et SLEEP.
Mis à 0 par un watchdog timer time out.

bit de zero :
mis à 1 si le résultat de l'ALU est égal à 0

bit de Power Down :
mis à 1 après une mise sous tension et par l'instruction CLRWDT
Mis à 0 par l'instruction SLEEP.

bit de Digit Carry :
bit de retenu du bit 3 vers le bit 4 au moment d'une addition ou d'une soustraction.

état des bits T0 et PD après un reset :

$\overline{TO}$	$\overline{PD}$	RESET was caused by
1	1	Power-up (POR)
u	u	$\overline{MCLR}$ reset (normal operation)(1)
1	0	$\overline{MCLR}$ wake-up reset (from SLEEP)
0	1	WDT reset (normal operation)
0	0	WDT wake-up reset (from SLEEP)

Legend: u = unchanged
Note 1: The $\overline{TO}$ and $\overline{PD}$ bits maintain their status (u) until a reset occurs. A low-pulse on the $\overline{MCLR}$ input does not change the $\overline{TO}$ and $\overline{PD}$ status bits.

Compléments sur les bits T0 et PD :
Les microcôntroleurs de la famille PIC 16C5X ne possèdent pas d'interruptions. Par contre, le watchdog, l'instruction SLEEP ou la mise à 0 de la broche MCLR provoquent un reset du composant. Donc la seule façon de connaître la cause d'un reset est de tester les bits T0 et PD du STATUS.

Rappel : un watchdog est un dispositif qui permet de surveiller le bon déroulement d'un programme.

2.6.3. FSR et INDF :

Ces deux registres sont utilisés pour l'adressage indirect (Cf figure 10). Le FSR pointe sur l'adresse du registre à lire (ou à écrire). INDF sert dans l'instruction à spécifier que l'adressage est indirect.

2.6.4. PC : Program Counter :

Le PC est codé sur 9 bits (PIC16C54/55), 10 bits (PIC16C56) ou 11 bits (PIC16C57/58). Le registre lisible par l'utilisateur est noté ***PCL*** (adresse 02h). L'utilisateur n'a donc accès qu'aux 8 bits de poids faible du PC. Pour l'utilisation de tables de données en mémoire programme ainsi que pour les instructions CALL et GOTO il faudra donc spécifier la page de destination (sauf pour les PIC16C54 et PIC 16C55 qui n'ont que 512 instructions).

Le principe de la construction du Program Counter en fonction du registre PCL et des bits de pages (PA1 et PA0) est donné aux figures 11, 12 et 13.

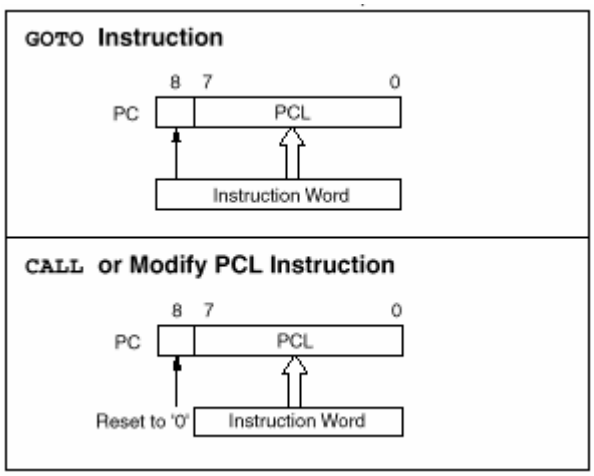


fig.11 : Construction de l'instruction de branchement pour les PIC16C54/55.

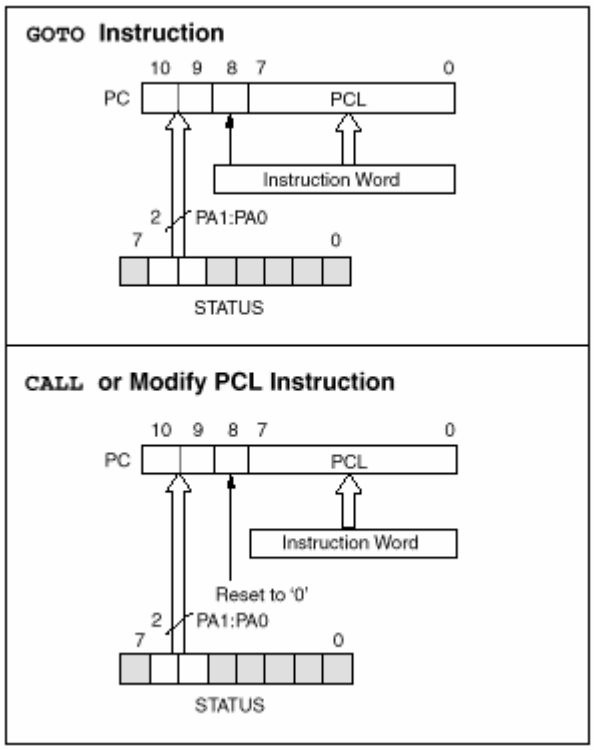


fig.12 : instruction de branchement pour le PIC16C56

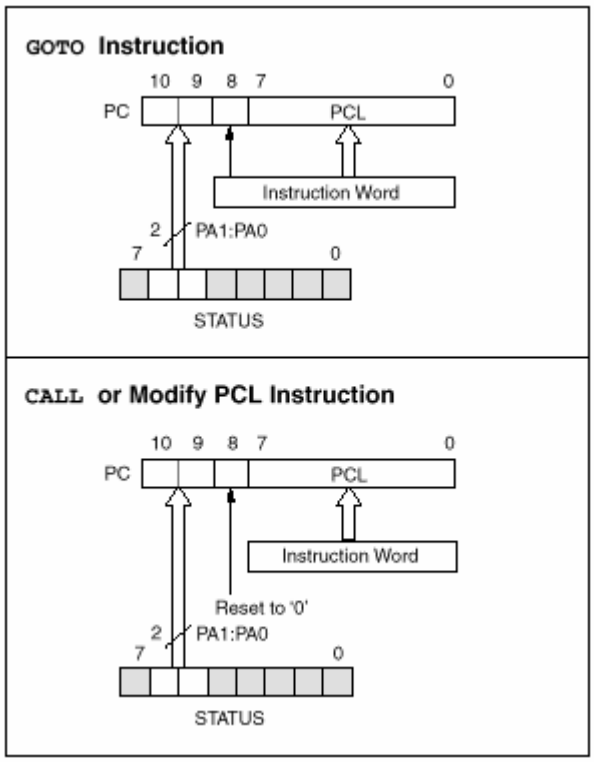


fig.13 : instruction de branchement pour les PIC16C57/58

Instruction GOTO :

Cette instruction fournit en opérande les 9 bis de branchement. Donc une instruction GOTO permet d'accéder à tout l'espace dans une page. Si l'on désire se brancher vers une autre page il faudra utiliser les registres de pages PA1 et PA0.

Exemple :

```
                ORG     0x00            ; exemple pour PIC 16C56/57/58
                BCF     STATUS, 6       ; sélection de la
                BSF     STATUS, 5       ; page 1 : PA1 PA0 = 0 1
                GOTO    SAUT_PAGE1      ; saut vers la page 1
RETOUR
                ORG     0x200           ; début de la page 1
SAUT_PAGE1      ;  instructions
                ; ..........
                BCF     STATUS, 6       ; sélection de la page 0
                BCF     STATUS, 5       ; PA1 PA0 = 0 0 pour un retour
                GOTO    RETOUR          ; vers la page 0.
```

Instruction CALL :

Cette instruction ne fournit en opérande que 8 bits (Cf chapitre 2.5.3). Le neuvième bit sera automatiquement mis à 0 (Cf figure 11, 12 et 13). ***Donc un CALL ne pourra atteindre que les 256 premières instructions de chaque page*** (Cf figure 14).

Exemple :

```
                ORG     0x00            ; exemple pour PIC 16C56/57/58
                BCF     STATUS, 6       ; sélection de la
                BSF     STATUS, 5       ; page 1 : PA1 PA0 = 0 1
                CALL    PROC_PAGE1      ; appel à la procédure en page 1 se trouvant dans les 256 1ères instructions
                BCF     STATUS, 6       ; sélection de la page 0
                BCF     STATUS, 5       ; PA1 PA0 = 0 0 pour la suite du programme
; remarque : à la fin de la procédure PROC_PAGE1 l'instruction retlw récupère la totalité du PC dans la pile
```

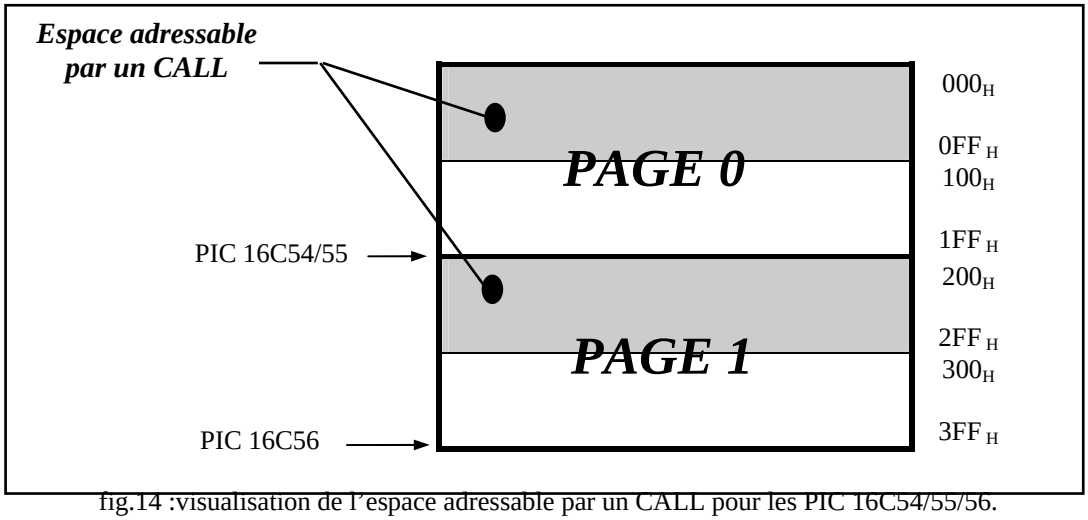


fig.14 :visualisation de l'espace adressable par un CALL pour les PIC 16C54/55/56.

Remarque : ***au reset tous les bits du PC sont mis à 1***. Donc la première instruction à placer sera :

cas du PIC 16C54/55	*cas du PIC 16C56*	*cas du PIC 16C57/58*
ORG 0x1FF ;	**ORG 0x3FF**	**ORG 0x7FF**
GOTO 0x000	**GOTO 0x000**	**GOTO 0x000**

2.6.5. Les ports d'entrée/sortie :

Les ports d'entrée/sortie, PORTA, PORTB et PORTC sont explicités en détail au chapitre 2.7.

2.7. Les registres d'entrées/sorties :

2.7.1. Présentation :

Il existe 2 types de registre :

- les registres de contrôle des ports ***notés TRIS A, TRIS B et TRIS C***. Ces registres ne font pas parties des registres du SFR. La seule façon d'écrire dans ces registres est l'instruction TRIS x (x étant l'adresse du PORT). Cette instruction envoie le contenu de W dans le registre TRIS x. Les bits mis à ***1*** dans ces registres mettent les broches correspondantes en haute impédance (voir figure 15) : ***broches configurées en entrée***. Les bits mis à ***0*** dans le registre TRIS x active les broches correspondantes : ***broches configurées en sortie***. La valeur alors écrite (0 ou 1) dans le bits du registre PORT x force (à 0 ou à 1) les broches correspondantes. Au reset les bits des registres TRIS x sont tous à 1 : Ports configurés en entrée.
- Les registres du SFR notés ***PORT A, PORT B et PORT C***. Pour le PORT A, seuls les 4 premiers bits sont utilisés. La lecture des 4 bits de poids supérieur renvoie un 0. Le PORT C pour les PIC 16C54/56 et 58 peut être utilisé comme un registre utilisateur.

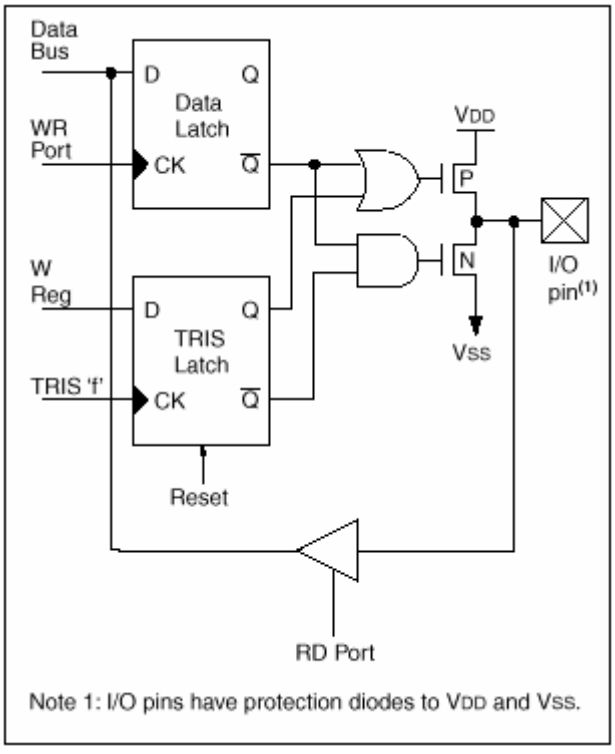


fig.15 :schéma d'un port d'entrée/sortie.

2.7.2. Exemple : commande d'un moteur pas à pas

Nous désirons commander un moteur pas à pas unipolaire à l'aide d'un PIC16C5X. Le schéma de principe est donné à la figure 16. L'appui sur le bouton poussoir noté ARRET devra stopper le moteur.

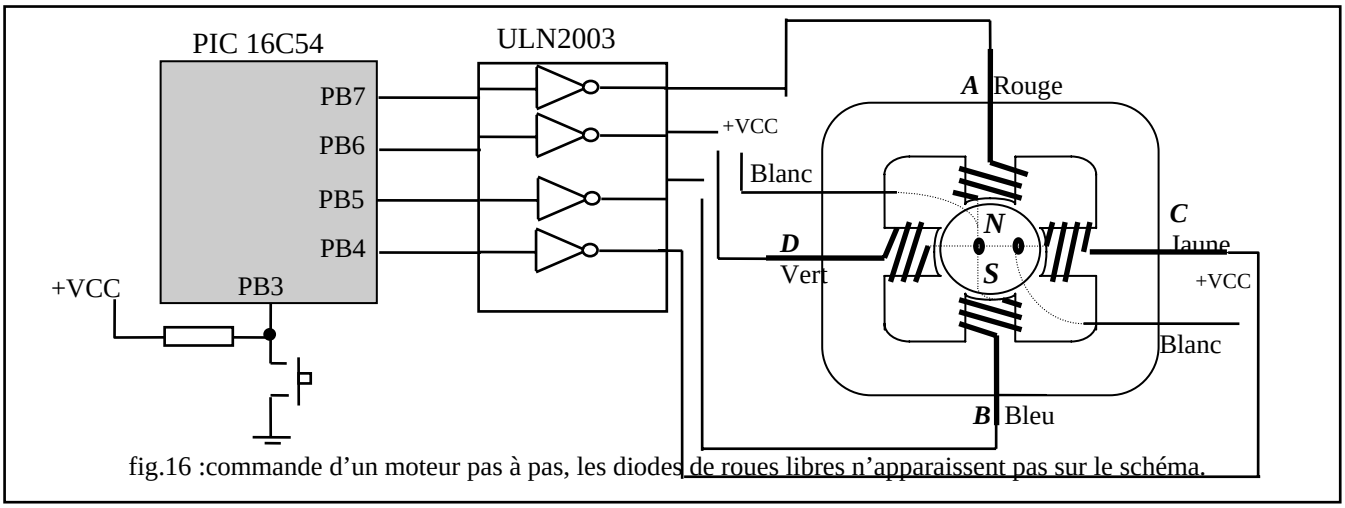


fig.16 :commande d'un moteur pas à pas, les diodes de roues libres n'apparaissent pas sur le schéma.

2.7.3. Rappels : le moteur pas à pas

Utilisations des moteurs pas à pas : Photocopieurs, machines à écrire, tables traçantes, applications faibles puissances.

Il existe 2 modes de fonctionnement pour un moteur pas à pas unipolaire :

- le mode monophase : on alimente un seul enroulement à la fois (si l'on ne dispose que d'une alimentation) ou les 2 enroulements opposés (si l'on dispose de 2 alimentations symétriques).
- le mode biphase : on alimente 2 enroulements adjacents (exemple A et D) si l'on ne dispose que d'une alimentation ou les 4 enroulements si l'on dispose de 2 alimentations symétriques. Dans ce cas le couple du moteur est maximal.

Les figures 17 et 18 montrent le positionnement du rotor en fonction de l'alimentation des phases dans le cas du mode monophase et du mode biphase. les phases notées X0 ne sont pas alimentées. Les phases notées X+ sont alimentées par un courant positif et les phases notées X- sont alimentées par un courant négatif (cas d'une alimentation symétrique) ou ne sont pas alimentées (cas d'un seule alimentation).

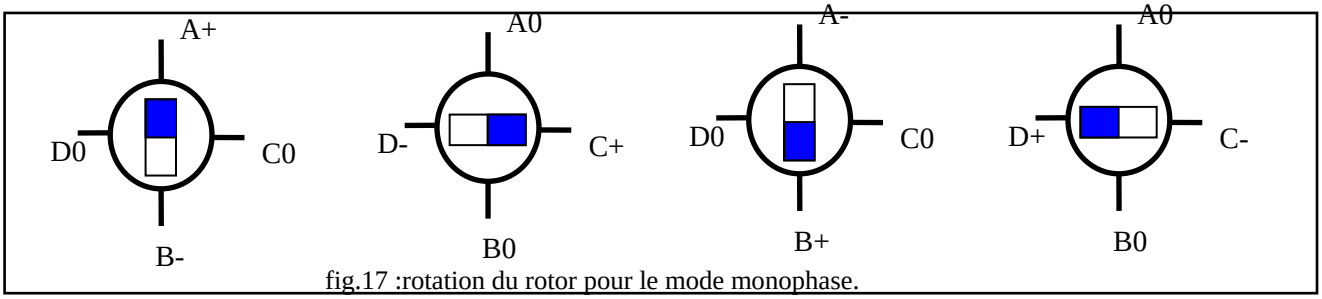


fig.17 :rotation du rotor pour le mode monophase.

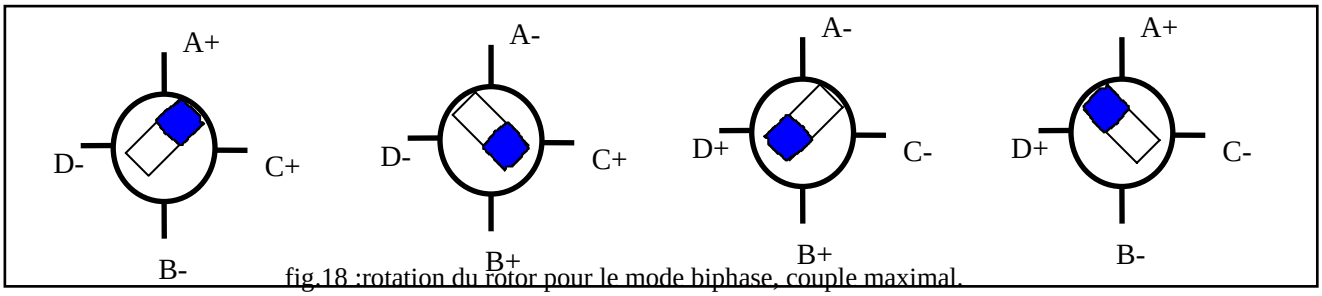
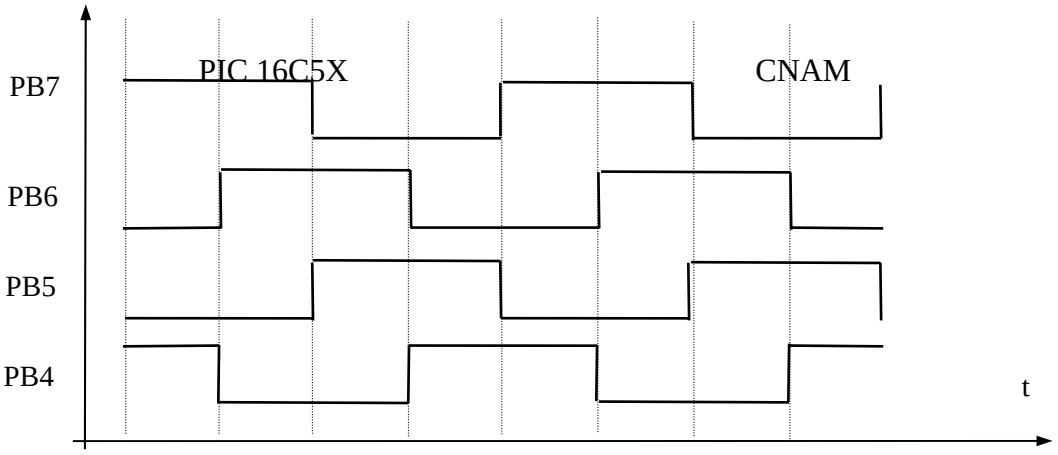


fig.18 :rotation du rotor pour le mode biphase, couple maximal.

2.7.4. Solution :

On désire obtenir les chronogrammes suivants :



Remarque : Quand les broches sont a 1, la sortie de l'amplificateur de courant (ULN2003) est à 0 et donc les phases sont alimentées. Le courant qui traversent une phase est limité par la résistance du bobinage.

```
;----------programme de commande du moteur pas à pas pour un PIC 16C54---------
; Ce programme est incomplé puisque il faudrait intégrer une boucle de temporisation pour limiter la fréquence des
signaux de commande du moteur. La fréquence d'oscillation choisi pour l'application est de 32 kHz, type LP.
                ORG         0x000
                MOVLW       B'10010000'     ; initialisation des bits <7..4> du PORTB
                MOVWF       PORTB
                MOVLW       0Fh             ; bits <7..4> du PORTB en sortie
                TRIS        PORTB           ; bits <3..0> du PORTB en entrée
RETOUR          MOVLW       B'10010000'     ; début de la séquence
                MOVWF       PORTB
                MOVLW       B'11000000'
                MOVWF       PORTB
                MOVLW       B'01100000'
                MOVWF       PORTB
                MOVLW       B'00110000'
                MOVWF       PORTB
                BTFSC       PORTB, 3        ; bit 3 du PORTB = 0 ?
                GOTO        RETOUR          ; non, retour
FIN                                         ; oui arret du moteur
                ORG         0x1FF
                GOTO        0x000           ; Initialisation du PC
```

2.7.5. Exercice complémentaire :

On désire afficher sur 8 LEDs connectées sur le PORT B la valeur d'un compteur incrémenté à chaque fois que l'utilisateur appuie sur un bouton poussoir connecté à la broche RA1. Le schéma de câblage est donné à la figure 19.

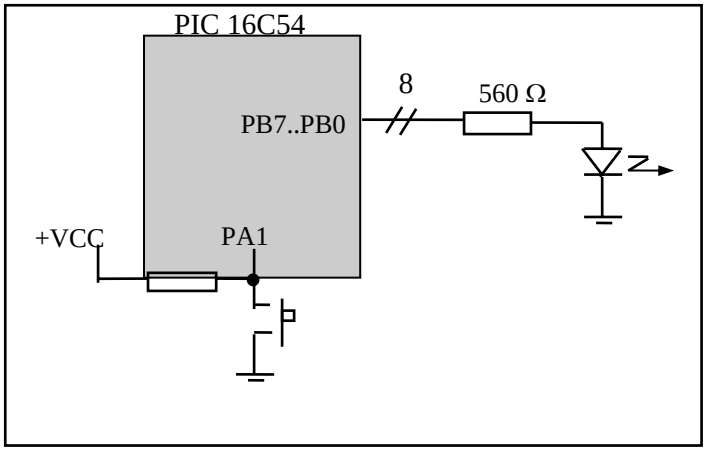


fig.19 : schéma de principe de l'exercice 1

Correction :

```
                list p = 16C54                  ; directive assembleur, génération de code pour PIC 16C54
                include « PIC16C5X.INC »        ; fichier de déclaration des registres
                ;
                COMPTEUR    EQU    10h          ; déclaration des variables utilisés
                ATTENTE     EQU    11h
                TEMPS_ATTEN EQU    FFh          ; temps d'attente anti-rebond T= TEMPS_ATTEN x (4xTosc)
                ;
                ORG         0
                MOVLW       0                   ; Initialisation du PORTB
                MOVWF       PORTB               ; PORTB configuré en sortie et les sorties forcées à 0
                TRIS        PORTB               ; PORTA configuré en entrée au RESET
                CLRF        COMPTEUR            ; Initialisation de compteur
BOUCLE          MOVLW       TEMPS_ATTEN ; initialisation de la variable ATTENTE
                MOVWF       ATTENTE
LIT_PA1         BTFSS       PORTA, 1            ; bit 1 du PORTA = 0 ?
                GOTO        INC_COMPT           ; oui, incrémenter le compteur et afficher
                GOTO        LIT_PA1             ; non, tester PA1
INC_COMPT       INCF        COMPTEUR
                MOVF        COMPTEUR,W
                MOVWF       PORTB
RELACHE         BTFSS       PORTA,1             ; bouton poussoir relaché ?
                GOTO        RELACHE             ; non, retester PA1
ANTI_REBOND     DECF        ATTENTE, F          ; attente de (4 x Tosc) x ATTENTE
                BTFSS       STATUS, Z
                GOTO        ANTI_REBOND
                GOTO        BOUCLE
;
                ORG         0x1FF
                GOTO        0x000               ; Initialisation du PC
```

2.8. Les registres OPTION et RTCC (ou TMR0) : gestion du Timer et du Watchdog

2.8.1. Le timer :

Le timer est composé d'un registre 8 bits noté RTCC ou TMR0 (adresse 01h du SFR) et d'un PRESCALER déclenchable sur front montant ou descendant (la programmation se fait par le registre OPTION, Cf figure 21). La figure 20 montre le fonctionnement du timer.

Remarque : il n'y a ***qu'un PRESCALER pour le timer et le watchdog***. Une attention particulière du programmeur devra être nécessaire pour bien gérer cette difficulté !

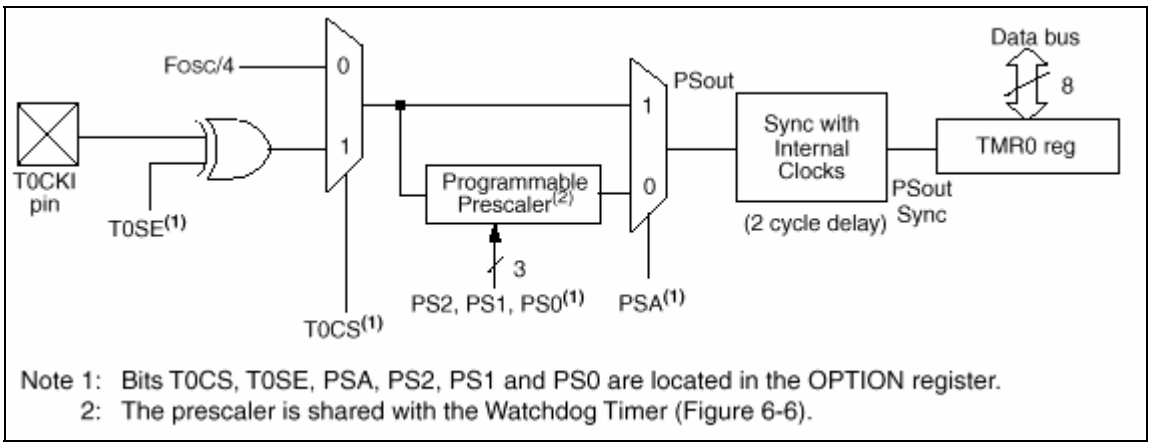
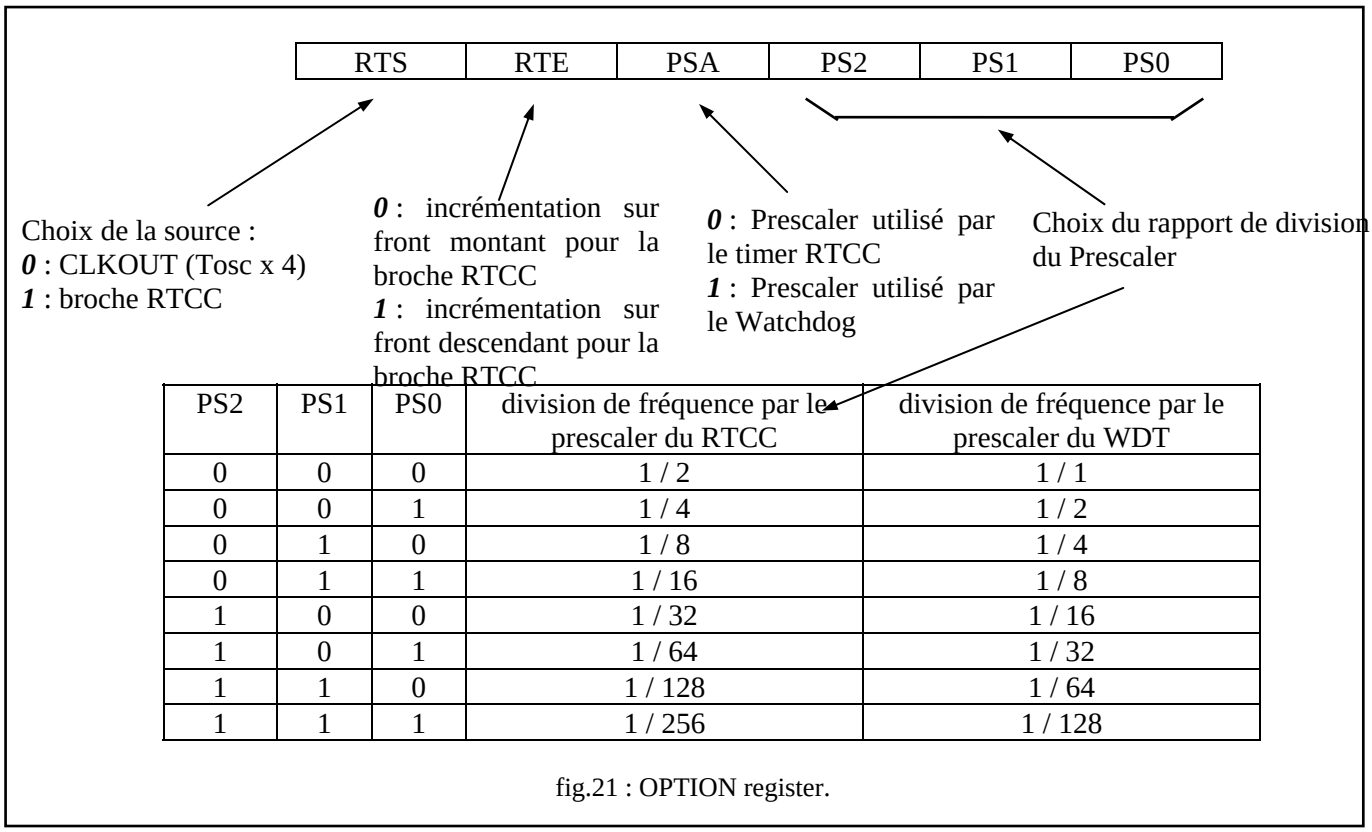


fig.20 :schéma simplifié du timer RTCC ouTMR0.



PS2	PS1	PS0	division de fréquence par le prescaler du RTCC	division de fréquence par le prescaler du WDT
0	0	0	1 / 2	1 / 1
0	0	1	1 / 4	1 / 2
0	1	0	1 / 8	1 / 4
0	1	1	1 / 16	1 / 8
1	0	0	1 / 32	1 / 16
1	0	1	1 / 64	1 / 32
1	1	0	1 / 128	1 / 64
1	1	1	1 / 256	1 / 128

fig.21 : OPTION register.

On accède au registre OPTION par l'instruction OPTION. Cette instruction transfère les 6 bits de poids faible du registre W dans le registre OPTION. Au reset, tous les bits du registre OPTION sont à 1.

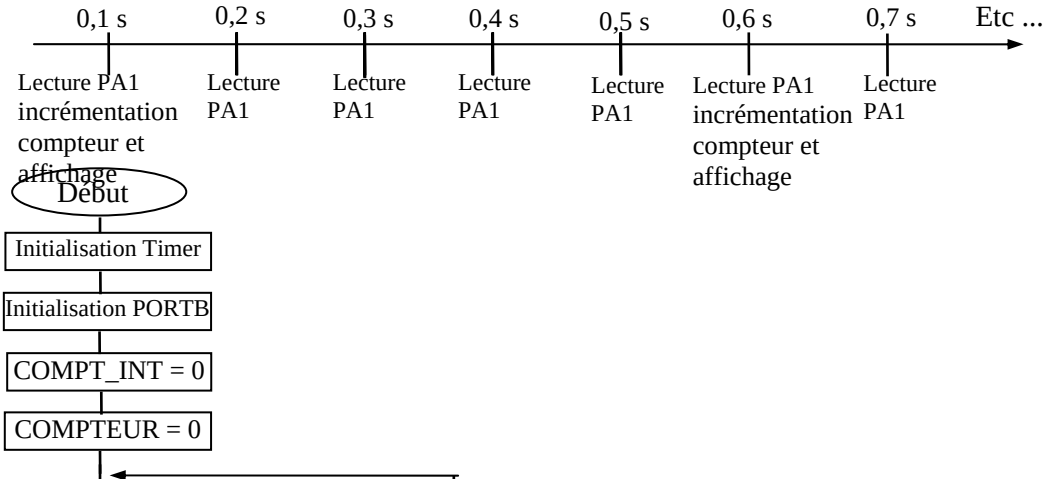
Pour expliquer le fonctionnement du timer, prenons deux exemples :

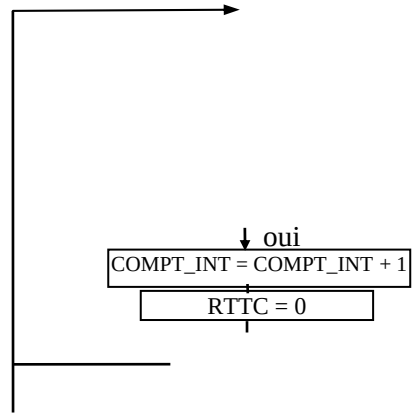
```
MOVLW   B'000x1xxx'     ; incrémentation de RTCC à chaque période
OPTON                   ; de l'horloge interne (pour chaque instruction)

MOVLW   B'00110000      ; incrémentation de RTCC tous les 2 fronts descendants
OPTION                  ; du signal sur la broche RTCC
```

Reprenons l'exercice du chapitre 2.9.5 et utilisons le timer pour générer une incrémentation du compteur toutes les 0,5 s. Un appui sur le bouton poussoir remet à 0 le compteur. De plus, on devra lire le bouton poussoir toutes les 0,1 s. Ainsi, on s'affranchit de l'éventuel problème des rebonds successifs au moment du relâchement du bouton poussoir. Nous utiliserons un oscillateur céramique (mode RC) qui oscille à 40 kHz. Le mode RC est choisit pour son coût très faible.

Le PRESCALER sera utilisé pour multiplier le temps de cycle par 256, ainsi le timer sera incrémenté toutes les 25,6ms.

On devra donc avoir le chronogramme suivant :






```
        list p = 16C54                  ; directive assembleur, génération de code pour PIC 16C54
        include « PIC16C5X.INC »        ; fichier de déclaration des registres
        COMPTEUR    EQU     10h         ; déclaration des variables utilisés
        COMPT_INT   EQU     11h
        TEST_BP     EQU     4           ; temps d'attente entre 2 lecture du bouton poussoir TEST_BP x T1 = 4 x 256 x 0,1 ms
        TEMP_AFF    EQU     5           ; temps d'attente entre 2 rafraîchissement de l'affichage T2 = 5 * T1 = 0,5ms
                ORG         0
                MOVLW       B'00010111'         ; incrémentation de RTCC toutes les 256 x 4 * Tosc = 25,6 ms
                OPTON
                CLRF        RTCC, F             ; initialisation de RTCC
                MOVLW       0                   ; Initialisation du PORTB
                MOVWF       PORTB
                TRIS        PORTB
                CLRF        COMPT_INT, F        ; Initialisation du compteur intermédiaire
                CLRF        COMPTEUR, F         ; initialisation du compteur
RETOUR          BTFSS       PORTA, 1            ; bit 1 du PORTA = 0 ?
                CLRF        COMPTEUR, F         ; oui, mise à 0 du compteur
WAIT            MOVLW       TEST_BP
                SUBWF       RTCC, W
                BTFSS       STATUS, Z       ; RTCC = TEST_BP = 4 ?
                GOTO        WAIT            ; non, attendre
                INCF        COMPT_INT, F    ; oui, COMPT_INT = COMPT_INT + 1
                CLRF        RTCC, F         ; RTCC = 0
                MOVLW       TEMP_AFF
                SUBWF       COMPT_INT, W
                BTFSS       STATUS, Z       ; COMPT_INT = TEMP_AFF = 5 ?
                GOTO        RETOUR          ; non, RETOUR
                INCF        COMPTEUR, F     ; oui, incrémentation de COMPTEUR et gestion de l'affichage
                MOVF        COMPTEUR, W
                MOVWF       PORTB           ; affichage toutes les 5 * 4 * 256 * 0,1 ms = 0,5 s
                CLRF        COMPT_INT, F
                GOTO        RETOUR
                ORG         0x1FF
                GOTO        0x000           ; Initialisation du PC
```

2.8.2. Le Watchdog :

Le Watchdog permet de vérifier le bon fonctionnement d'un programme. Ainsi ce compteur devra être remis à 0 à intervalle régulier par le programme. Si pour une raison particulière, le programme ne répondait plus, le Watchdog se trouverait en dépassement (n'ayant pas été remis à 0) et donc provoquerait un reset du microcôntroleur.

Le Watchdog Timer (noté WDT) est un oscillateur RC interne indépendant de l'oscillateur de l'horloge du PIC. Ce qui veut dire que le WDT fonctionne même si l'horloge du PIC est arrêtée par l'instruction SLEEP.

Le WDT est activé ou non au moment du téléchargment du programme (menu de l'assembleur MPASM). C'est un bit de configuration WDTE qui permet l'activation ou pas du WDT.

Le WDT crée un signal de dépassement (time out period) toutes les 18 ms (si le prescaler n'est pas utilisé, bit PSA du registre OPTION égal à 0) ou toutes les 18ms x valeur du prescaler (max = 128 x 18 ms = 2,3 s).

Le signal de dépassement du WDT provoque un reset du microcôntroleur. Le schéma de principe est donné à la figure 22.

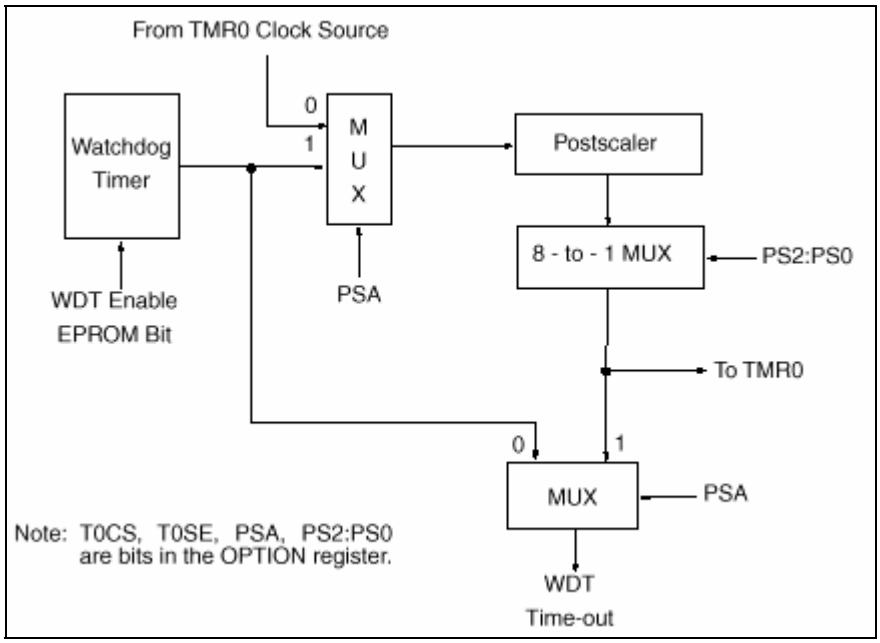


fig.22 : schéma du wathdog timer.

L'instruction **CLRWDT** remet à 0 le WDT et le Prescaler. Donc après un **CLRWDT**, il y aura un reset du microcôntroleur au bout de 18 ms si le prescaler n'est pas utilisé ou de 18 ms x valeur du prescaler si le bit PSA du registre OPTION est égal à 1 (la valeur du prescaler est défini par les bits PS2, PS1, PS0 du registre OPTION).

L'instruction **SLEEP** permet l'arrêt de l'horloge du microcôntroleur pour une économie d'énergie. Cette instruction met à 0 le WDT et le prescaler. Si le WDT est activé (bit de configuration WDTE = 1), celui-ci générera un signal de dépassement (temps défini par le programmeur dans le registre OPTION) et donc un reset du microcôntroleur (reveil du PIC).

Les causes d'un reset sont : (Cf figure 23)

- mise sous tension du composant
- dépassement (timer out watchdog) du WDT
- dépassement du WDT après une instruction SLEEP, réveil du microcôntroleur
- mise à 0 de la broche MCLR (Master Clear)

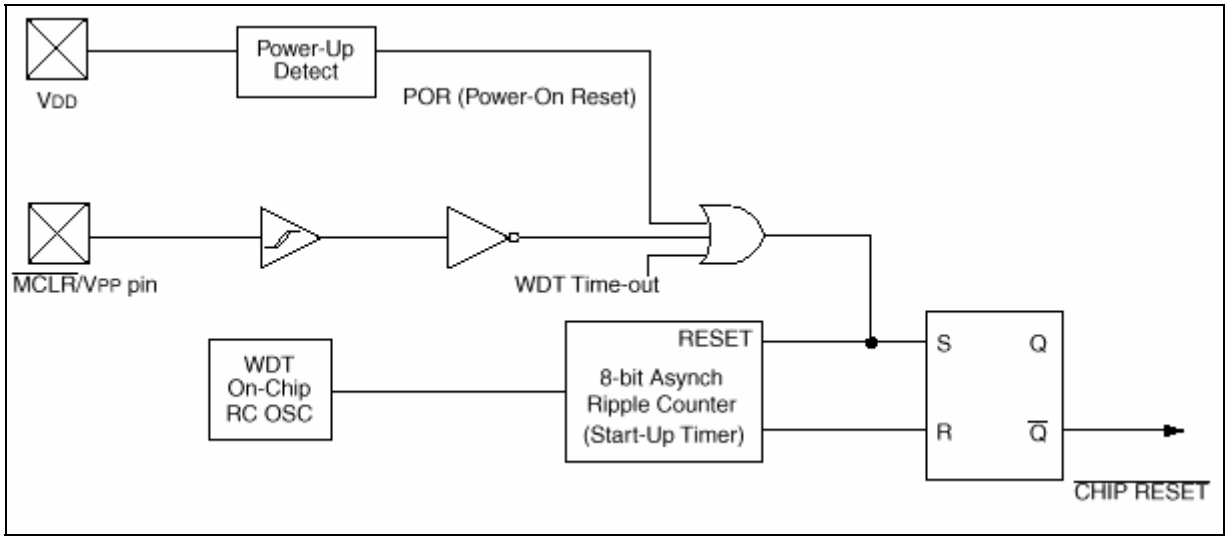


fig. 23 : causes d'un reset du PIC.

2.9. Les bits de configuration :

Ces bits de configuration (le menu de téléchargement de MPASM demande à l'utilisateur de configurer ces bits) permettent : (Cf figure 24)

- de placer un code de protection contre la lecture (bit de sécurité)
- d'activer ou de désactiver le watchdog timer
- de choisir le type d'oscillateur utilisé (RC ou quartz), Cf figure 4.

CP	CP	CP	CP	CP	CP	CP	CP	CP	WDTE	FOSC1	FOSC0
bit11	10	9	8	7	6	5	4	3	2	1	bit0

Register: CONFIG
Address(1): FFFh

bit 11-3: **CP:** Code protection bits
1 = Code protection off
0 = Code protection on

bit 2: **WDTE:** Watchdog timer enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0: **FOSC1:FOSC0:** Oscillator selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

fig.24 : bits de configuration.

3. Applications diverses :

3.1. Exemples :

Si A > B alors X = X + 1
sinon X = X - 1

```
        MOVF   A, W
        SUBWF  B, W
        BTFSC  STATUS, C      ; si B ≥ A alors C = 0
        GOTO   ALORS          ; si B < A alors INCF X
        DECF   X, F           ; sinon  DECF X
        GOTO   FIN
ALORS   INCF   X, F
FIN
```

Cas Où NUMERO =
1 : Faire ACTION1

```
        MOVF    NUMERO, W     ; NUMERO est un
registre
        GOTO    TABLE         ; qui contient la valeur
```

2 : Faire ACTION2
3 : Faire ACTION3
Fin Cas Où

Faire
.....
.....
Tant Que X ≠ 0

```
TABLE  ADDWF  PC , F
       GOTO   ACTION1
       GOTO   ACTION2
       GOTO   ACTION3

FAIRE
       ........
       ........
       MOVLW  0
       SUBWF  X, W
       BTFSS  STATUS, Z
       GOTO   FAIRE
```

3.2. Génération d'une sinusoïde :

On désire générer une sinusoïde avec un PIC16C5X en utilisant le montage fait au Chapitre 7 du polycopié MCS 51.

```
; Ce programme n'est pas complet. Il permet juste de se familiariser avec la façon de programmer les PIC16C5X
            CLRF   X
BOUCLE      MOVF   X, W
            CALL   SINUS  ; PORTB = SINUS (X)
            MOVWF  PORTB
            INCF   X
            CALL   ATTENTE        ; attente entre 2 échantillons
            GOTO   BOUCLE
SINUS       ADDWF  PC
            RETLW  128
            RETLW  131
            RETLW  134
            RETLW  137
            RETLW  141
            ; ...etc
```

3.3. Gestion d'un clavier :

Cet exemple est tiré de « l'Application Note 529 » fourni par Microchip. Avant de s'intéresser à l'exemple, nous allons faire le point sur les différentes façons de lire un clavier.

3.3.1. Lecture de boutons poussoirs :

La technique la plus simple consiste à connecter directement chaque bouton poussoir (ou touche du clavier) à une broche du port du microcôntroleur. Cette technique est illustrée par les figures 25 et 26.

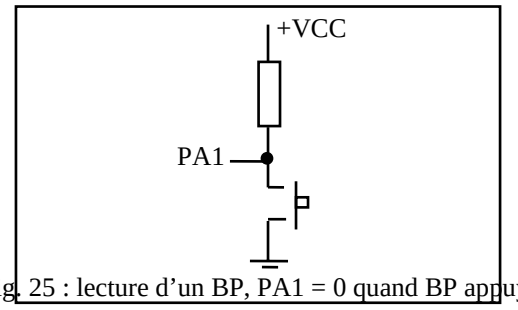


fig. 25 : lecture d'un BP, PA1 = 0 quand BP appuyé

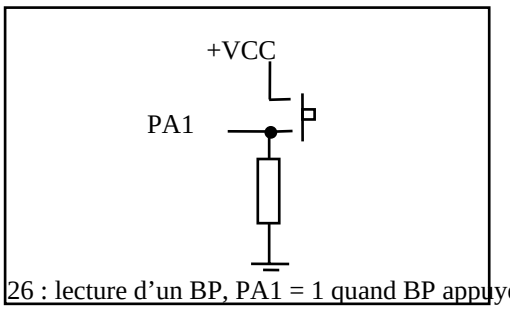


fig. 26 : lecture d'un BP, PA1 = 1 quand BP appuyé

Il est évident que cette technique ne sera utilisé que si le nombre de touches ou de boutons poussoirs est faible (inférieur à 5 ou 6 touches).

3.3.2. Gestion d'un clavier en connexion matricielle en lecture :

Par une connexion matricielle, on arrive à diminuer le nombres de broches utilisées pour la lecture du clavier : pour 9 touches, on utilisera 3 + 3 broches et pour 16 touches, 8 broches. Cette technique est donnée à la figure 27. Toutes les broches du port sont en lecture.

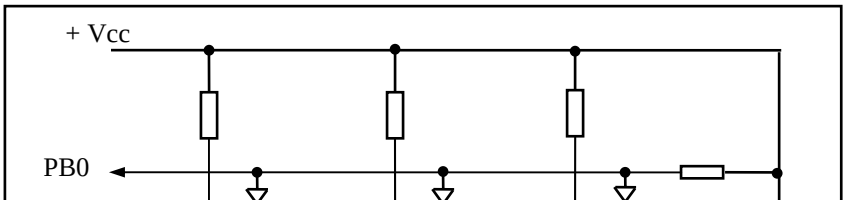


fig.27 : clavier 9 touches avec lecture matricielle

Remarque : si l'utilisateur appuie sur 2 touches en même temps, le microcôntroleur est capable de lire ces 2 touches, par contre le nombre de diodes utilisées est important.

3.3.3. Gestion d'un clavier en connexion matricielle en lecture-écriture :

La gestion du clavier se fait par 3 écritures successives d'un 0 sur les lignes (pendant que les 2 autres lignes sont à 1) et une lecture sur les colonnes. Ainsi l'appui sur la touche 1 est détecté lorsque PB0 est mis à 0 (PB1 et PB2 forcé à 1) la broche PB3 passe à 0.

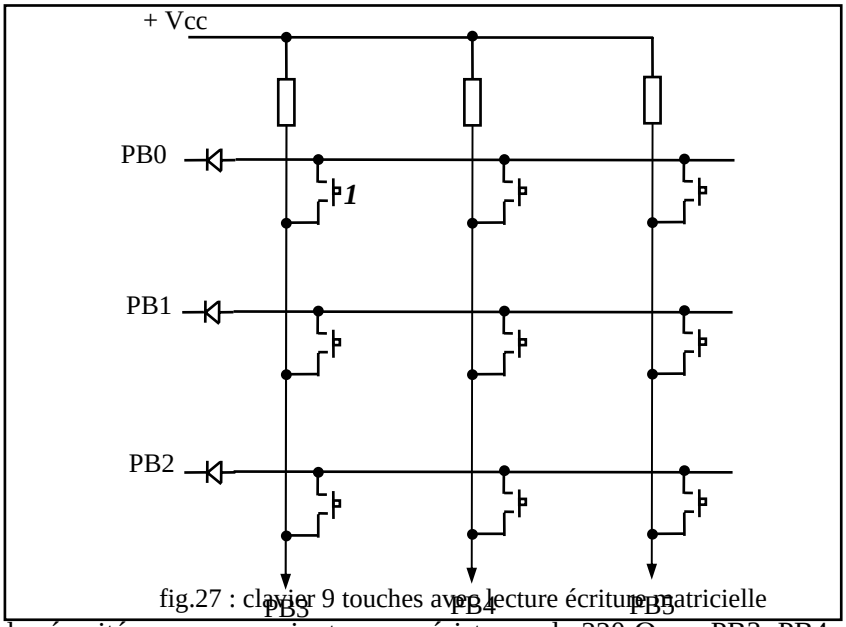


fig.27 : clavier 9 touches avec lecture écriture matricielle

Pour plus de sécurité, on pourra rajouter une résistance de 220 Ω sur PB3, PB4 et PB5 et cela pour éviter un court-circuit entre les broches des colonnes et les broches des lignes si par inadvertance le programmeur forçait les broches PB3, PB4 et PB5 en sortie.

On remarque que cette technique est moins coûteuse que la précédente, par contre elle demande un peu plus de code pour la mise en œuvre.

3.3.4. Gestion d'un clavier 4x4 et d'un afficheur à LED : AN 529

Le but de cette application est de faire une horloge avec remise à l'heure possible à partir d'un clavier. De part le faible nombre de broches des ports d'entrée/sortie, il a fallu utiliser les mêmes broches pour lire le clavier et pour écrire vers l'afficheur. Le port PB sera placé en sortie et commandera l'afficheur par multiplexage (broches

RA0, RA1, RA2 et RA3). Ainsi toutes les 5 ms, le port PA activera un des 4 afficheurs. Toutes les 20 ms les broches RB4, RB5, RB6 et RB7 seront configurées en entrée et l'on lira le clavier (Le chronogramme de la mise en œuvre est donné à la figure 29). Cette lecture qui durera un peu plus de 0,17 ms ne perturbera pas l'affichage. Les résistances R8-R11 et R12-R15 forment un pont diviseur qui divisent la tension VCC par 10 ce qui, lors de l'appui sur une touche, fait passer le potentiel sur les broches configurées en entrées à 0,5V. Lorsque les broches RB4, RB5, RB6 et RB7 sont configurées en sortie, l'appui sur une touche ne vient pas perturber l'afficheur. Ceci est du au fait que les résistances R8-R15 sont bien supérieures aux résistances R0-R7 de gestion de l'afficheur.

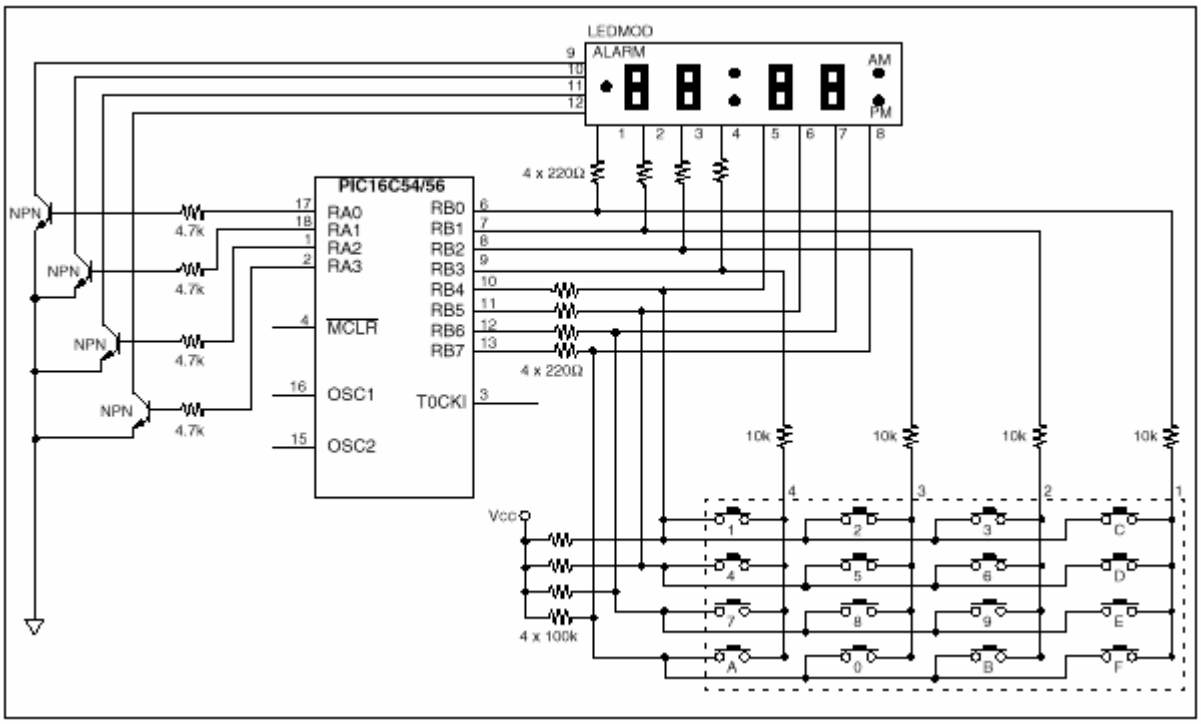
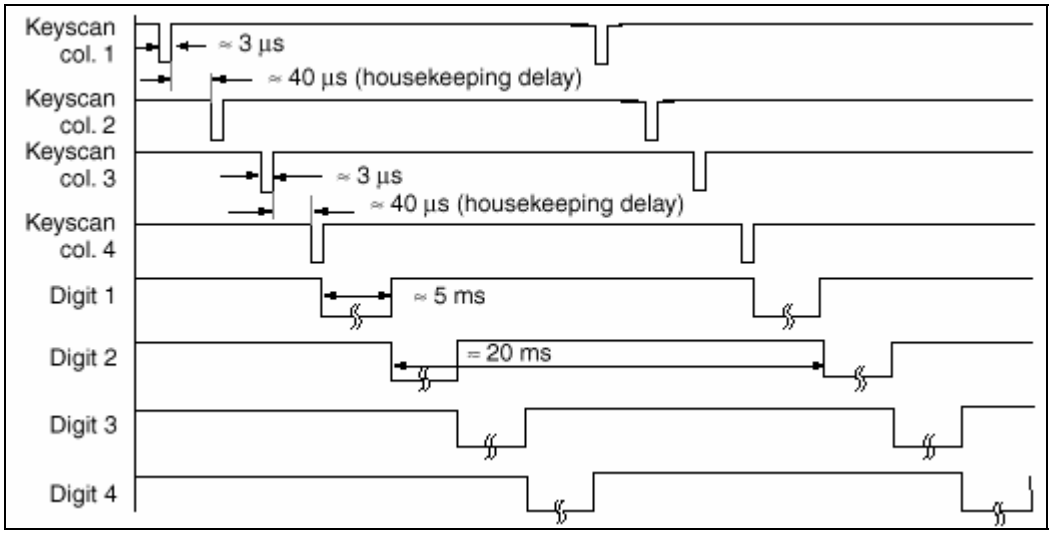


fig. 29 : horloge avec gestion d'un clavier



On utilisera un quartz de 4,096 MHz. Une instruction sera donc exécutée toutes les µs. Donc si l'on utilise le TIMER avec une division par le PRESCALER de 32 et en initialisant le TIMER à 96 on aura un dépassement toutes les (256 - 96) x 32 = 5 ms. Ces 5 ms seront utilisées pour gérer l'affichage et pour incrémenter un compteur qui permettra de compter les secondes, les minutes et les heures.

```
; ce programme incomplet permet de lire le clavier et de placer dans la variable RESULT le résultat de la lecture
; RESULT devra être compris entre 0 et 15
                MOVLW       B'11110000'
                TRIS        PORTB       ; PB7..4 en entrée et PB3..0 en sortie
                MOVLW       B'11110111'
                MOVWF       TEMP        ; TEMP : variable temporaire utilisée pour la lecture du clavier
SKP1            MOVF        TEMP, W
                MOVWF       PORTB       ; écriture d'un 0 sur la colonne à scanner
                MOVF        PORTB, W    ; lecture de la ligne
```

```
                ANDLW       B'11110000'     ; masquage des bits de poids faible
                XORLW       B'11110000'     ; si touche appuyé W <> 0
                BTFSS       STATUS, Z       ; touche appuyée ?
                GOTO        DET_TOUCHE      ; oui :
SKP3            BSF         STATUS, C       ; non :
                RRF         TEMP, F         ; décalage de TEMP, pour l'écriture vers la colonne suivante
                BTFSC       STATUS, C       ; 4 colonnes écrites, c.a.d. lecture du clavier finie ?
                GOTO        SKP1            ; non :
                CLRF        NOUV_TOUCHE     ; drapeau de touche appuyée
SKP2            CLRF        PORTB           ; fin de la lecture du clavier
                MOVLW       B'00000000'
                TRIS        PORTB           ; Port B remis en sortie pour l'affichage
                GOTO        FIN_LECTURE
DET_TOUCHE
; ce prog est appelé seulement si une touche a été apuyée. Il renvoie le résultat de la lecture dans NOUV_TOUCHE
; exemple :si la première touche de la première colonne touche 1 a été appuyée alors NOUV_TOUCHE = B'01110111'
                MOVF        PORTB, W        ; lecture à nouveau du PORTB
                IORLW       B'00001111'     ;
                ANDWF       TEMP, W         ;
                MOVWF       NOUV_TOUCHE     ;
                GOTO        SKP2
FIN_LECTURE
; ce prog doit en fonction de la valeur de NOUV_TOUCHE en déduire la valeur à afficher sur un afficheur 7 segment
                MOVF        NOUV_TOUCHE, W ;
                BTFSC       STATUS, Z       ; si pas de nouvelle touche alors
                GOTO        FIN_FIN         ; pas de changement
                ANDLW       B'00001111'     ; masquage de la ligne
                MOVWF       TOUCHE          ; lecture de la touche courante
                MOVLW       3
                MOVWF       TEMP            ; initialisation de TEMP
LECT_COL        BSF         STATUS, C
                RRF         TOUCHE, F       ; bit de poids faible dans C
                BTFSS       STATUS, C       ; quelle ligne appuyée ?
                GOTO        LECT_LIGNE
                DECFSZ      TEMP, F         ; TEMP contient le numéro de la colonne ou une touche a été appuyé
                GOTO        LECT_COL
LECT_LIGNE      MOVF        TEMP, W
                MOVWF       RESULT          ; RESULT = 0, 1, 2 ou 3 en fonction de la touche appuyée
                MOVWF       TOUCHE          ; TOUCHE permet de lire la ligne
                MOVLW       3
                MOVWF       TEMP            ; initialisation de TEMP
LECT            BSF         STATUS, C
                RLF         TOUCHE, F       ; bit de poids fort dans C
                BTFSS       STATUS, C       ; quelle ligne appuyée ?
                GOTO        RESULTAT
                DECFSZ      TEMP, F         ; TEMP contient le numéro de la colonne ou une touche a été appuyé
                GOTO        LECT
RESULTAT        MOVF        TEMP, W
                ADDWF       PC, F
                GOTO        GET147A
                GOTO        GET2580
                GOTO        GET369B
                GOTO        GETCDEF
GET147A         MOVLW       0
                GOTO        RETOUR
GET2580         MOLW        4
                GOTO        RETOUR
GET369B         MOVLW       8
                GOTO        RETOUR
GETCDEF         MOVLW       D'12'
                GOTO        RETOUR
RETOUR          ADDWF       RESULT, F       ;
; RESULT =  0  1  2  3
;           4  5  6  7
;           8  9  A  B
;           C  D  E  F ... en fonction de la touche tapée
FIN_FIN
```

3.4. Asservissement d'un moteur à courant continu :

On désire asservir la vitesse d'un moteur à courant continu à l'aide d'un PIC 16C54. Pour cela on dispose d'un C.A.N. série à approximations successives (ADC0801), d'un opto-coupleur (4N25) et d'une dynamo tachimétrique. Le schéma de principe est donné à la figure 30.

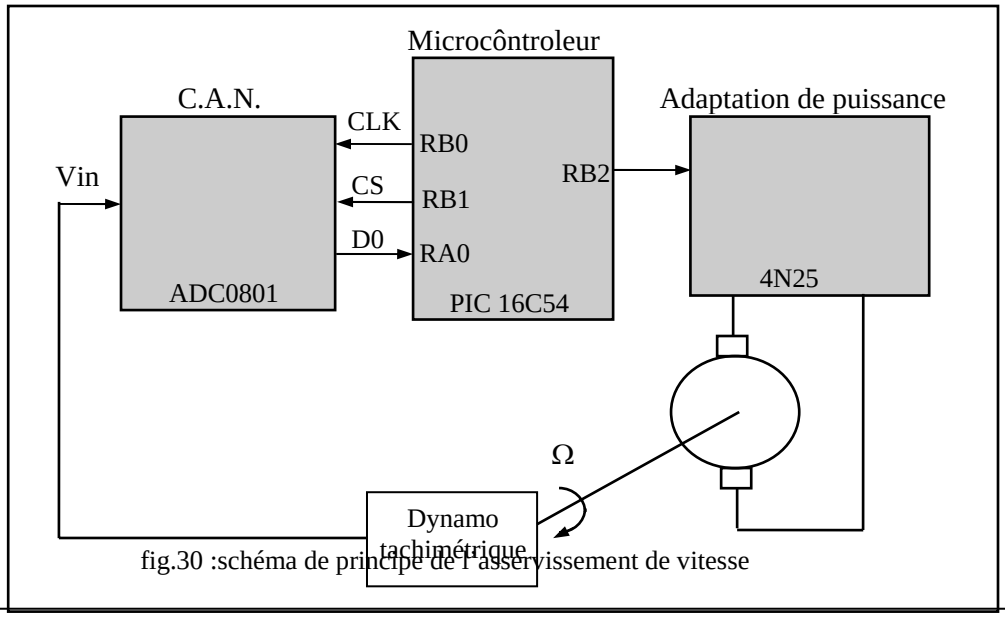


fig.30 :schéma de principe de l'asservissement de vitesse

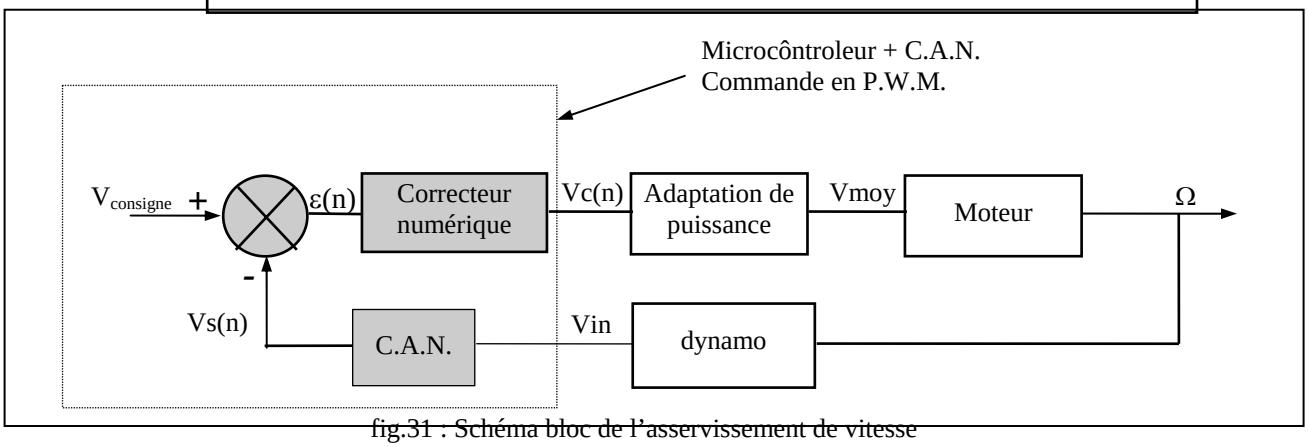


fig.31 : Schéma bloc de l'asservissement de vitesse

Un asservissement est une loi fonction de la différence entre la valeur voulue ($V_{consigne}$) et la valeur réelle (Vlu). Cette loi est codée dans le correcteur numérique. Le schéma bloc de l'asservissement est donné à la figure 31. Il existe 2 façons de trouver cette loi :

* en utilisant les résultats connus pour les asservissements analogiques (types P.I. ou P.I.D.),
- en utilisant les transformées en z, on utilise alors les résultats connus pour la régulation numérique.

Ce chapitre ne traitera pas de la recherche de la loi du correcteur numérique, mais de la programmation de l'asservissement dans le PIC. Pour plus de détail, on se référera au cours d'automatique sur les asservissements.

Pour lire la tension en sortie de la dynamo (noté Vin), on utilise un C.A.N.(l'ADC0801 donné en ANNEXE). Pour commander le moteur, il aurait donc fallu un C.N.A. Ors ce C.N.A. n'apparaît pas dans le montage. C'est normal puisque l'on commande le moteur en P.W.M. ou en M.L.I.(Modulation à Largeur d'Impulsion). La figure 32 donne un exemple des signaux en entrée du moteur. On prendra comme période d'échantillonnage Te = 25ms = τ/3 avec τ constante de temps électromécanique donnée par la réponse en boucle ouverte du moteur (fourni en ANNEXE). Pour que le moteur moyenne la valeur fournie par le PIC, il faut que la fréquence de la PWM soit suffisamment grande. On prendra un Ts = 5 ms soit une fréquence de découpage de 2 kHz.

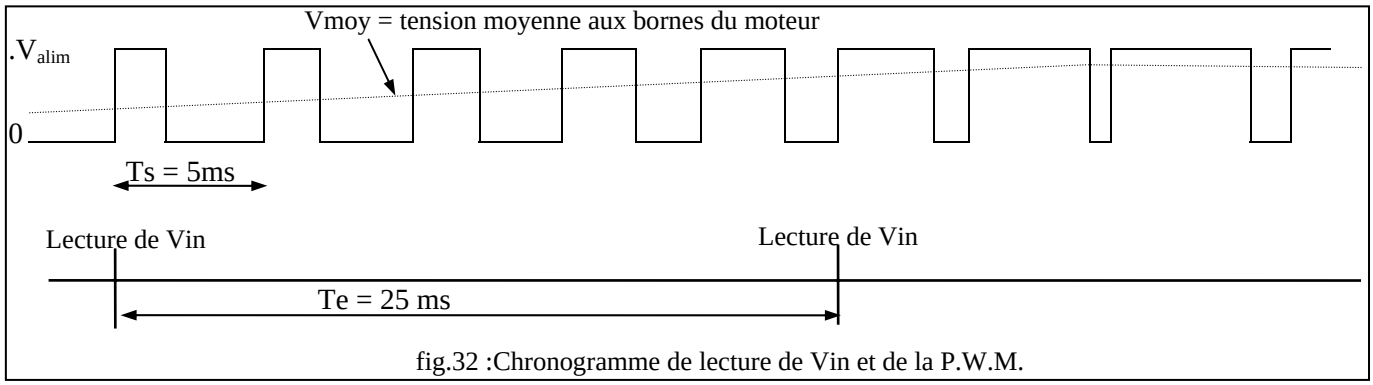
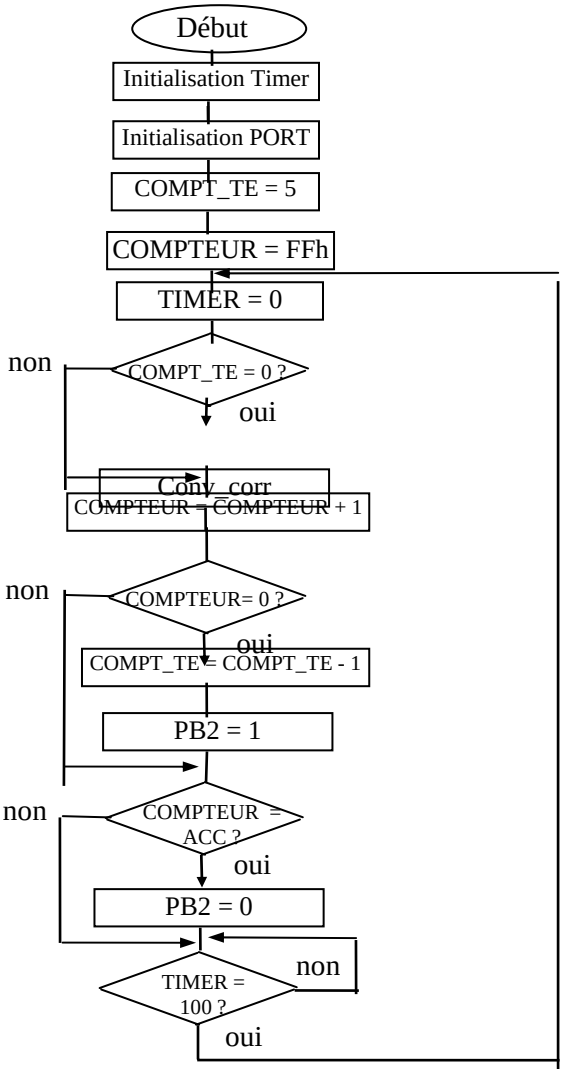
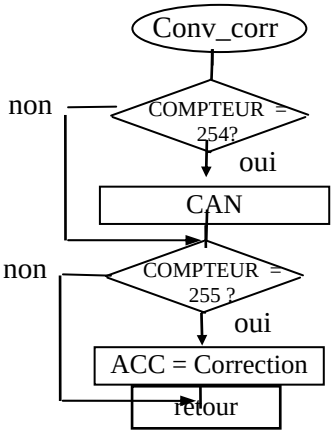


fig.32 :Chronogramme de lecture de Vin et de la P.W.M.

Si l'on prend une fréquence d'horloge Fh = 20 MHz, on a un temps de cycle de 200 ns. Il faudra 100 instructions x 256 instructions pour gérer une période Ts de la PWM.

Exercice : donner l'algorithme de l'asservissement et donner la procédure de gestion du C.A.N.






```
;programme de gestion du C.A.N. ADC0801
                CLRF        VSN, F          ; résultat de la conversion VSN = 0
                MOVLW       9
                MOVF        COMPTEUR, F     ; passage 9 fois dans la boucle retour
                BCF         PORTB, CS       ; mise à 0 de RB1
                BCF         PORTB, CLK      ; mise à 0 de RB0
RETOUR          BSF         PORTB, CLK      ; mise à 1 de RB0
                MOVF        PORTA, W        ; lecture port A
                MOVLW       1
                ANDWF       TEMP, F         ; masquage des bits 7..1
                NOP                         ; rapport cyclique du CLK = 0,5
                NOP
                BCF         PORTB, CLK      ; mise à 0 de RB0
                RRF         TEMP, F         ; C = RA0
                RLF         VSN, F          ;
                DECF        COMPTEUR, F     ; COMPTEUR = COMPTEUR -1
                BTFSS       STATUS, Z       ; COMPTEUR = 0 ?
                GOTO        RETOUR          ; NON : retour
                BSF         PORTB, CS       ; arrêt de la conversion, résultat dans VSN
                RET
```