Le microcontrôleur PIC16F84

Plan

I. FAMILLE PIC DE MICROCHIP

➔ **Introduction générale**

Une PIC est un microcontrôleur, c'est à dire une unité de traitement de l'information de type microprocesseur à laquelle on a ajouté des périphériques internes;

La dénomination PIC est sous copyright de Microchip;

Les PICs sont des composants dits RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit;

I. FAMILLE PIC DE MICROCHIP

➔ **Les différents familles des PICs**

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);
- La famille High-End, qui utilise des mots de 16 bits.

I. FAMILLE PIC DE MICROCHIP

➔ **Identification d'une PIC**

16 F 84 -04

Catégorie du PIC: PIC Mid-Range

Indique une mémoire de type Flash

Identification du PIC

fréquence d'horloge maximale (4 Mhz)

Le PIC existe aussi en deux versions:

- PIC16F 84 : pour une utilisation dans une gamme d'alimentation classique (4.5 à 5.5V).
- PIC16LF84: pour une gamme étendue de l'alimentation (2 à 6V).

II. ORGANISATION DU 16F84

➔ **Description Générale**

Le PIC16F84 est un microcontrôleur 8 bits qui offre un très bon rapport qualité/prix.

Ses principales caractéristiques:

- seulement 35 mots d'instructions
- vitesse jusqu'à 10Mhz
- 4 sources d'interruption
- 1000 cycles effacement/écriture possible de la mémoire programme flash
- 1K mots mémoire programme Flash
- 68 octets de données RAM
- 64 octets de données EEPROM

II. ORGANISATION DU 16F84

Principales caractéristiques: (suite)

- 13 Entrée/Sortie
- 1 Timer/Compteur
- 4 Sources d'oscillateur sélectionnable
- Mode sleep (pour une faible consommation)
- Programmation par ISP (In Serial Programming)

II. ORGANISATION DU 16F84

Avantages :

- Système à faible coût
- Encombrement réduit
- Meilleure fiabilité
- Mise en œuvre facilitée

Adaptés :

- Aux grandes séries
- Aux systèmes embarqués

Inconvénients :

- Performance des périphériques réduite
- Inadaptés à la gestion de gros systèmes
- Utilisation simultanée de tous les périphériques impossible

II. ORGANISATION DU 16F84

Domaines d'utilisation :

- Systèmes embarqués
- Petits systèmes économiques
- Systèmes de commande à faible diffusion (prototypes…)
- Tout système ne nécessitant pas des ressources importantes.

II. ORGANISATION DU 16F84

Choix d'un microcontrôleur :

Deux critères essentiels :

Puissance du processeur

- Format des mots traités
- Fréquence d'horloge
- Architecture interne.
- Jeu d'instructions et adressages, nombre de registres.
- Adaptation aux langages évolués.

Périphériques intégrés

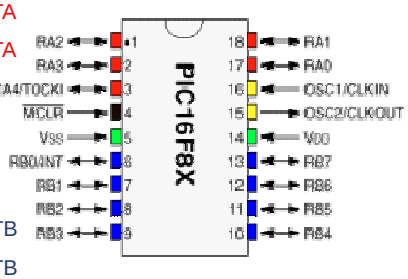
- Mémoire interne (RAM, ROM, EEPROM…)
- Nombre de lignes d'E/S
- Nombre de compteurs, précision, …
- Périphériques spécialisés

II. ORGANISATION DU 16F84

➔ **Structure externe**

✔ Le PIC16F84 est logé dans un boîtier 18 broches DIL ou SOIC (version CMS).

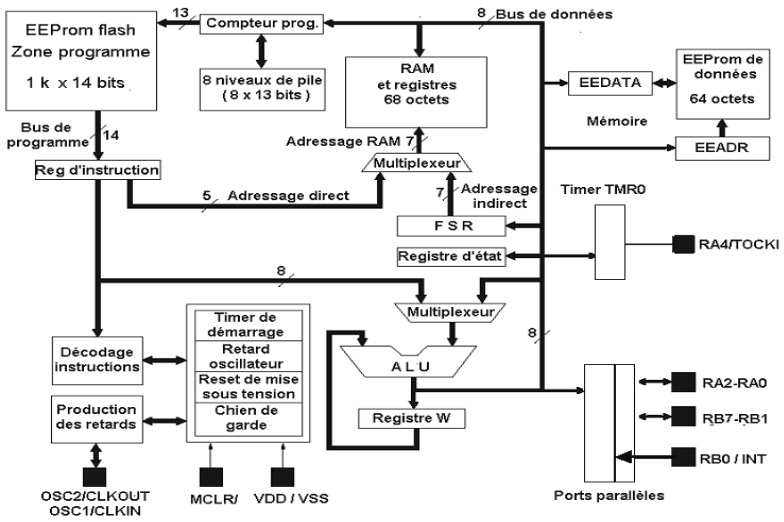
PIN 1: 3ème bit du PORTA
PIN 2: 4ème bit du PORTA
PIN 3: 5ème bit du PORTA / TOCK
PIN 4: RESET
PIN 5: GND
PIN 6: 1er bit du PORTB
PIN 7: 2ème bit du PORTB
PIN 8: 3ème bit du PORTB
PIN 9: 4ème bit du PORTB



PIN 18: 2ème bit du PORTA
PIN 17: 1er bit du PORTA
PIN 16: OSCILLATEUR
PIN15: OSCILLATEUR 2
PIN 14: VCC
PIN 13: 8ème bit du PORTB
PIN 12: 7ème bit du PORTB
PIN 11: 6ème bit du PORTB
PIN 10: 5ème bit du PORTB

II. ORGANISATION DU 16F84

➔ **Structure interne**



II. ORGANISATION DU 16F84

➔ **Organisation de la mémoire**

La mémoire du PIC16F84 est divisée en deux parties distinctes:

*la mémoire programme;
*la mémoire de donnée.

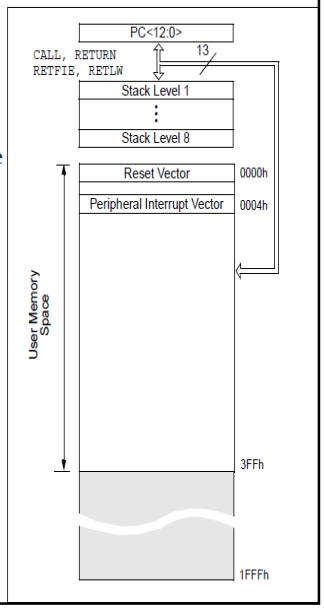
✓ Le premier contient les instructions du programme ainsi que les vecteurs RESET et INTERRUPTION.
✓ Dans la mémoire de donnée se trouve tout les registres nécessaires pour la configuration et l'utilisation des périphériques internes ainsi que 68 octets de RAM utilisateurs.

II. ORGANISATION DU 16F84

La mémoire programme

La mémoire programme est constituée de 1K mots de 14 bits.
RESET ⇒ le PIC16F84 commence à l'adresse 0000H (Vecteur RESET).
Interruption ⇒ le microcontrôleur va à l'adresse 0004H (Vecteur d'interruption).
On trouve donc en mémoire programme un mot de 14 bits:

Exemple: movlw 10 110000 00001010



II. ORGANISATION DU 16F84

La mémoire RAM

La mémoire RAM est organisée en 2 banques pour la 16F84.La RAM est subdivisée de plus en deux partie Dans chacune des banques nous avons des cases mémoires spéciales appelées Registres spéciaux et 68 octets de cases mémoires libre pour notre utilisation .

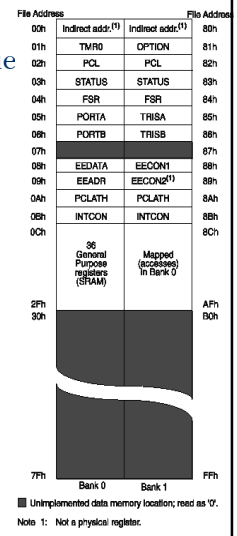
-En page 0, des registres fondamentaux

-En page 1, des registres associés

-Les adresses 00 et 07 ne contiennent aucune mémoire .

01 TMRO: contenu du timer 8 bits, il peut être Incrémenté par l'horloge (fosc/4) ou RA4.

02 PCL: 8 bits du poids faible du PC, les 5 bits Du poids fort sont dans PCLATH .

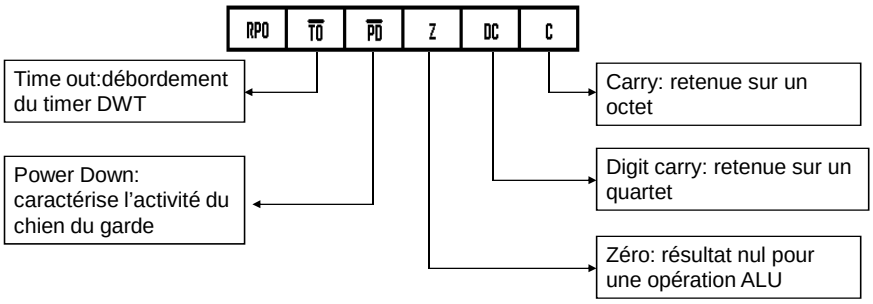


II. ORGANISATION DU 16F84

03 STATUS: registre d'état contenant:

- 5 bits, drapeaux caractérisant le résultat de l' opération ALU; **(lecture)**

- 1 bit de sélection de page RP0 (0: page 0). **(lecture /écriture)**



II. ORGANISATION DU 16F84

04 FSR: contient l'adresse d'un autre registre (adressage indirect).
05 PORTA et 06 PORTB: registres de données des ports parallèles.
08 EEDATA: contient un octet lu ou a écrire dans l'EEPROM de données.
09 EEADR: contient l'adresse de la donnée lue ou écrite dans l'EEPROM .
0B INTCON: contrôle des 4 interruptions.

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Masques :
GIE : (Global Interrupt Enable) : masque global d'inter.
EEIE : (EEProm Interrupt Enable)
T0IE : (Timer 0 Interrupt Enable)
INTE: (Interrupt Enable) Entrée d'inter sur broche RB0/INT
RBIE: (RB Interrupt Enable) Entrées d'inter sur broches RB4-RB7.

II. ORGANISATION DU 16F84

Drapeaux :
T0IF : (Timer 0 Interrupt Flag) débordement du timer
INTF: (Interrupt Flag) interruption provoquée par la broche RB0/INT
RBIF: (RB Interrupt Flag) interruption provoquée par les broches RB4-RB7.
81 OPTION_REG: 8 bits (tous a 1 au reset) affectant le comportement des E /S et timers.

II. ORGANISATION DU 16F84

$\overline{\text{RBPU}}$	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0

RBPU/: (RB Pull Up) Résistances de tirage à Vdd des entrées du port B.
INTEDG: (Interrupt Edge) Front actif sur RB0 (1 pour front montant).
RTS: (Real Timer Source) Signal alimentant timer0 : 0 pour horloge interne, 1 pour RA4/T0CLK
RTE (Real Timer Edge) front actif du signal timer (0 pour front montant).
PSA (Prescaler assignment) 0 pour Timer 0 et 1 pour chien de garde WDT.
PS2..0 (Prescaler 210) Valeur du Diviseur de fréquence pour les timers.

II. ORGANISATION DU 16F84

85 TRISA: Direction des données pour le port A : 0 pour sortir et 1 pour entrer
86 TRISB: Direction des données pour le port B : 0 pour sortir et 1 pour entrer
88 EECON1: Contrôle le comportement de l'EEPROM de données.

EEIF	WRERR	WREN	WR	RD

EEIF: (EEProm Interrupt Flag) passe à 1 quand l'écriture est terminée.
WRERR: (Write Error) 1 si erreur d'écriture.
WREN: (Write Enable) : 0 pour interdire l'écriture en EEProm de données.
WR: (Write) 1 pour écrire une donnée. Bit remis automatiquement à 0
RD: (Read) : 1 pour lire une donnée. Bit remis automatiquement à 0

II. ORGANISATION DU 16F84

89 EECON2: Registre de sécurité d'écriture en EEPROM de données.

Une donnée ne peut être écrite qu'après avoir écrit successivement 0x55 et 0xAA dans ce registre :

```
movlw Adresse
movwf EEADR        ;définition de l'adresse

movlw Donnee
movwf EEDATA       ;définition de la donnée

movlw Ox55         ;)
movwf EECON2       ;) sécurité
movlw OxAA         ;)
movwf EECON2       ;)
bsf EECON1,WR      ; écriture
```

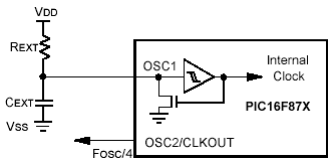
III. Les éléments de base du PIC 16F876/877

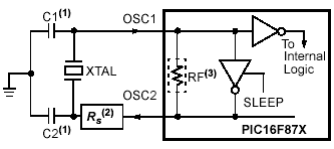
L'Horloge

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.





III. Les éléments de base du PIC 16F876/877

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.

III. Les éléments de base du PIC 16F876/877

Organisation de la mémoire RAM

L'espace mémoire RAM adressable « 16F876 » 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

III. Les éléments de base du PIC 16F876/877

Accès à la RAM par 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

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.

IV. 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.

IV. Les instructions du 16F876/877

➔ **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)

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

IV. Les instructions du 16F876/877

➔ **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.

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)

IV. Les instructions du 16F876/877

➔ **Les instructions opérant sur une constante**

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

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

IV. Les instructions du 16F876/877

➔ **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.

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

IV. Les instructions du 16F876/877

➔ **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

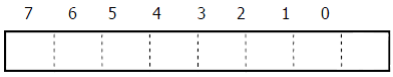
XORWF 35h,0 ou XORWF 35h,w

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

IV. Les instructions du 16F876/877

➔ **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

IV. Les instructions du 16F876/877

➔ **Les instruction MOVWF et MOVF**

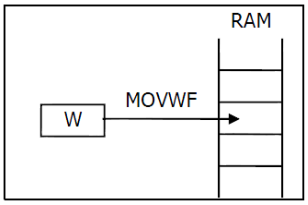
Ce sont les instructions les plus utilisées, MOVWF permet de copier l'accumulateur W dans un registre (SFR ou GPR):

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



IV. Les instructions du 16F876/877

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

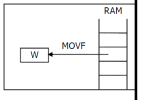
MOVF STATUS,0 ;

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

IV. Les instructions du 16F876/877

➔ **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
comf 70h,f ; après btfss, le programme continue ici si Z=1
suite du programme
suite du programme …

IV. Les instructions du 16F876/877

➔ **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.

IV. Les instructions du 16F876/877

➔ **L'instruction goto**

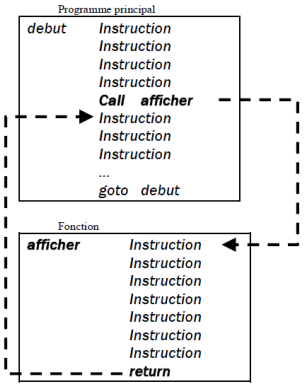
Permet de transférer l'exécution à une autre position du programme repérée par une étiquette (label)

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

IV. Les instructions du 16F876/877

➔ **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

IV. Les instructions du 16F876/877

Mnémoniques	Descriptions
ADDWF f,d	Additionne W et f
ANDWF f,d	AND de W avec f
CLRF f	Efface f
CLRW	Efface W
COMF f,d	Effectue le Complément de f
DECF f,d	Décrémente f
DECFSZ f,d	Décrémente f, passe si 0
INCF f,d	Incrémente f
INCFSZ f,d	Incrémente f, passe si 0
IORWF f,d	OR Inclusif de W avec f
MOVF f,d	Déplace f
MOVWF f	Déplace W dans f
NOP	Pas d´opération (No Opération)
RLF f,d	Rotation gauche à travers la Retenue (Carry)
RRF f,d	Rotation droite à travers la Retenue (Carry)
SUBWF f,d	Soustrait W de f
SWAPF f,d	Bascule f sur lui même
XORWF f,d	XOR de W avec f

IV. Les instructions du 16F876/877

Opérations sur les Bit

Mnémoniques	Descriptions
BCF f,b	Met à 0 le bit b de f
BSF f,b	Met à 1 le bit b de f
BTFSC f,b	Test le bit b de f, passe si à 0
BTFSS f,b	Test le bit b de f, passe si à 1

Opérations Littérales et de Contrôles

Mnémoniques	Descriptions
ANDLW k	AND d´un Littéral avec W
CALL k	Appel d´une sous-fonction
CLRWDT k	Efface le 'Watchdog Timer'
GOTO k	Branchement inconditionnel
IORLW k	OR Inclusif d´un Littéral avec W
MOVLW k	Place un Littéral dans W
OPTION k	Charge le registre OPTION
RETLW k	Retourne au programme principal, place un Littéral dans W
SLEEP	Se met en mode Stand by
TRIS f	Charge le registre TRIS
XORLW k	XOR d´un Littéral avec/vers W

V. Les ports d'E/S

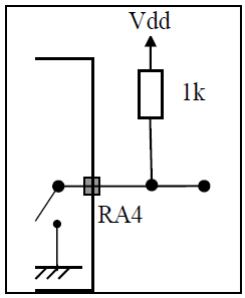
➔ **Le port d' E/S PORTA**

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

La broche RA4

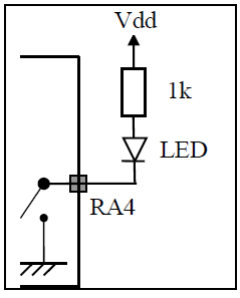
En entrée, la broche RA4 peut être utilisée soit comme E/S numérique normale, soit comme entrée horloge pour le Timer TMR0

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



V. Les ports d'E/S

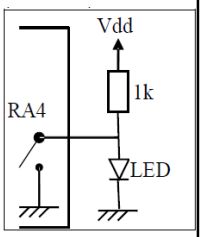
Si on veut utiliser RA4 pour allumer une LED, on peut utiliser le schéma de la Fig suivante. Il faut juste remarquer que la logique est inversée, si on envoie 0 sur RA4, l'interrupteur se ferme et la LED s'allume. Si on envoie 1, l'interrupteur s'ouvre et la LED s'éteint



V. Les ports d'E/S

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, 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



V. Les ports d'E/S

Les autres broches de PORTA

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 « 16F876 – 16F877 »

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

V. Les ports d'E/S

➔ **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.

V. Les ports d'E/S

➔ **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 entrée 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.

V. Les ports d'E/S

➔ **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 tous les processeurs 16F87X, il est disponible sur le 16F877, le 16F874 et le 16F871
- PORTD peut être utilisé dans un mode particulier appelé parallel slave port, pour cela il faut placer le bit PSPMODE (bit 4) 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 PSPMODE de TRISE à 0

V. Les ports d'E/S

➔ **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 tous les processeurs 16F87X, il est disponible sur le 16F877, le 16F874 et le 16F871
- 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 PSPMODE 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 broches de PORTE sont configurées comme entrées analogiques.

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

VI. LES MODES D'ADRESSAGE

la 16F84 utilise 3 modes d'adressages:

1. Adressage littéral ou immédiat:

2. Adressage direct:

3. Adressage indirect:
Cet adressage fait aux registres INDF et FSR.
INDF indirect File registre d'adresse 0x00, ce registre n'existe pas vraiment, ce n'est qu'un procédé d'accès particulier à FSR utilisé par la PIC pour des raisons de facilité de construction électronique interne .

FSR est à l'adresse 0x04 dans les 2 banques.
Pour modifier la case mémoire pointée, on modifie FSR
Pour connaître l'adresse de la case pointée, on accède à FSR
Pour accéder au contenu de la case pointée, on accède via INDF

VI. LES MODES D'ADRESSAGE

➔ **L'adressage littéral ou immédiat**

Avec l' ADRESSAGE IMMEDIAT ou ADRESSAGE LITTERAL, vous pouvez dire : 'je mets 100F en poche'. La valeur fait IMMEDIATement partie de la phrase. J'ai donné LITTERALlement la valeur concernée. Pas besoin d'un autre renseignement.

Exemple
movlw 0x55 ; charger la valeur 0x55 dans W

VI. LES MODES D'ADRESSAGE

➔ **L'adressage direct**

Avec l' ADRESSAGE DIRECT, vous pouvez dire : je vais mettre le contenu du coffre numéro 10 dans ma poche. Ici, l'emplacement contenant la valeur utile est donné DIRECTement dans la phrase. Mais il faut d'abord aller ouvrir le coffre pour savoir ce que l'on va effectivement mettre en poche. On ne met donc pas en poche le numéro du coffre, mais ce qu'il contient. Pour faire l'analogie avec les syntaxes précédentes, je peux dire que je mets (coffre 10) dans ma poche.

Exemple
movf 0x10 , W ; charger le contenu de l'emplacement 0x10 dans W

VI. LES MODES D'ADRESSAGE

➔ **L'adressage indirect**

Avec l' ADRESSAGE INDIRECT , vous pouvez dire :
Le préposé du guichet numéro 3 va me donner le numéro du coffre qui contient la somme que je vais mettre en poche.
Ici, vous obtenez le numéro du coffre INDIRECTement par le préposé au guichet.
Vous devez donc allez demander à ce préposé qu'il vous donne le numéro du coffre que vous irez ouvrir pour prendre l'argent. On ne met donc en poche, ni le numéro du préposé, ni le numéro du coffre que celui-ci va vous donner. Il y a donc 2 opérations préalables avant de connaître la somme que vous empocherez.

Cette adressage fait appel à 2 registres, dont un est particulier, car il n'existe pas vraiment.

VI. LES MODES D'ADRESSAGE

Les registres FSR et INDF

INDF signifie INDirect File, c'est le fameux registre de l'adresse 0x00. Ce registre n'existe pas vraiment, ce n'est qu'un procédé d'accès particulier à FSR utilisé par le PIC pour des raisons de facilité de construction électronique interne.
Le registre FSR est à l'adresse 0x04 dans les 2 banques. Il n'est donc pas nécessaire de changer de banque pour y accéder, quelle que soit la banque en cours d'utilisation.
L'adressage indirect est un peu particulier sur les PICS, puisque c'est toujours à la même adresse que se trouvera l'adresse de destination. En somme, on peut dire qu'il n'y a qu'un seul préposé (FSR) dans notre banque. ***Comment cela se passe-t-il ?***

VI. LES MODES D'ADRESSAGE

Premièrement, nous devons écrire l'adresse pointée dans le registre FSR.
Ensuite, nous accédons à cette adresse pointée par le registre INDF.
On peut donc dire que INDF est en fait le registre FSR utilisé pour accéder à la case mémoire. Donc, quand on veut modifier la case mémoire pointée, on modifie FSR, quand on veut connaître l'adresse de la case pointée, on accède également à FSR. Si on veut accéder au CONTENU de la case pointée, on accède via INDF.

Remarque: Le contenu du registre FSR pointe sur une adresse en 8 bits. Or, sur certains PICs, la zone RAM contient 4 banques (16F876). L'adresse complète est donc une adresse sur 9 bits.
L'adresse complète est obtenue, en adressage DIRECT, par l'ajout des bits 7 et 8 sous forme de RP0¨et RP1 et par l'ajout du bit IRP dans le cas de l'adressage INDIRECT. Veillez donc à toujours laisser IRP (dans le registre STATUS) et RP1 à 0 pour assurer la portabilité de votre programme.

VI. LES MODES D'ADRESSAGE

Exemple

```
movlw 0x50          ; chargeons une valeur quelconque
movwf mavariable ; et plaçons-la dans la variable « mavariable »
movlw mavariable ; on charge l'ADRESSE de mavariable, par
                    ; exemple, dans les leçons précédentes, c'était
                    ; 0x0e. (W) = 0x0E
movwf FSR           ; on place l'adresse de destination dans FSR.
                    ; on dira que FSR POINTE sur mavariable
movf INDF,w       ; charger le CONTENU de INDF dans W.
```

LE CONTENU DE INDF EST TRADUIT PAR LE PIC COMME ETANT LE CONTENU DE L'EMPLACEMENT MEMOIRE POINTE PAR FSR (W) = 0X50

VI. LES MODES D'ADRESSAGE

Exemples

movlw mavariable

C'est de l'adressage immédiat ou littéral ; donc on charge la VALEUR de mavariable.

movf mavariable , w

Cette fois, c'est de l'adressage DIRECT, donc, on va à l'adresse mavariable voir ce qu'il y a à l'intérieur. On y trouve le CONTENU de mavariable.

movf INDF , w

Maintenant, c'est de l'adressage INDIRECT. Ce mode d'adressage se reconnaît immédiatement par l'utilisation du registre INDF. Le PIC va voir dans le registre FSR, et lit l'adresse contenue. Elle va ensuite dans l'emplacement visé, et lit le CONTENU.

movf FSR , w

Ceci est un piège. C'est en effet de l'adressage DIRECT. On placera donc dans (W) le CONTENU du registre FSR.

VII. LES INTERRUPTIONS SUR LES PICs

➔ **Qu'est-ce qu'une interruption ?**

Votre programme se déroule normalement.
- Survient un événement spécifique.
- Le programme principal est interrompu (donc, subit une INTERRUPTION), et va traiter l'événement, avant de reprendre le programme principal à l'endroit où il avait été interrompu.

L'interruption est donc une RUPTURE DE SEQUENCE ASYNCHRONE, c'est-à-dire non synchronisée avec le déroulement normal du programme.
Vous voyez ici l'opposition avec les ruptures de séquences synchrones, provoquées par le programme lui-même (goto, call, btfss…).

VII. LES INTERRUPTIONS SUR LES PICs

➔ **Mécanisme d'interruption sur les PICs**

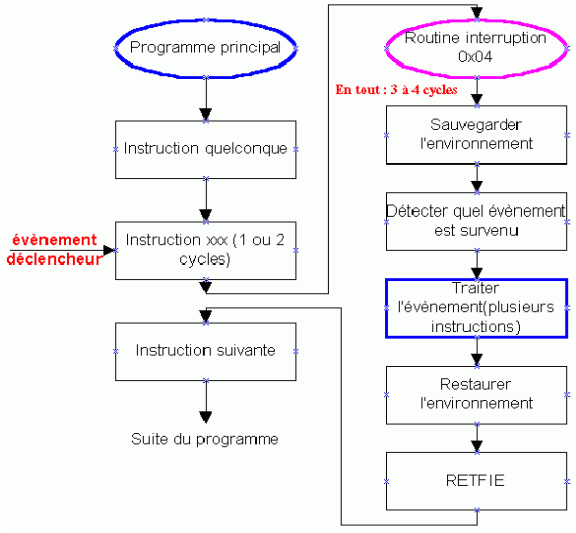
Tout d'abord, l'adresse de début de toute interruption est fixe. Il s'agit toujours de l'adresse 0x04.
* Les PICs en se connectant à cette adresse, ne sauvent rien à part le contenu du PC, qui servira à connaître l'adresse du retour de l'interruption.
* Le contenu du PC est sauvé sur la pile interne (8 niveaux). Donc, on dispose que de 7 niveaux d'imbrication pour les sous-programmes.
* Le temps de réaction d'une interruption est calculé de la manière suivante : le cycle courant de l'instruction est terminé, le flag d'interruption est lu au début du cycle suivant. Celui-ci est achevé, puis le processeur s'arrête un cycle pour charger l'adresse 0x04 dans PC. Le processeur se connecte alors à l'adresse 0x04 où il lui faudra un cycle supplémentaire pour charger l'instruction à exécuter. Le temps mort total sera donc compris entre 3 et 4 cycles.

VII. LES INTERRUPTIONS SUR LES PICs

* Une interruption ne peut pas être interrompue par une autre interruption. Les interruptions sont donc invalidées automatiquement lors du saut à l'adresse 0x04 par l'effacement du bit GIE.
* Les interruptions sont remises en service automatiquement lors du retour de l'interruption. L'instruction RETFIE agit donc exactement comme l'instruction RETURN, mais elle repositionne en même temps le bit GIE.

VII. LES INTERRUPTIONS SUR LES PICs

Les interruptions sur les PICs



VII. LES INTERRUPTIONS SUR LES PICs

➔ **Les sources d'interruptions du 16F84**

La 16F84 ne dispose que de 4 sources d'interruptions:

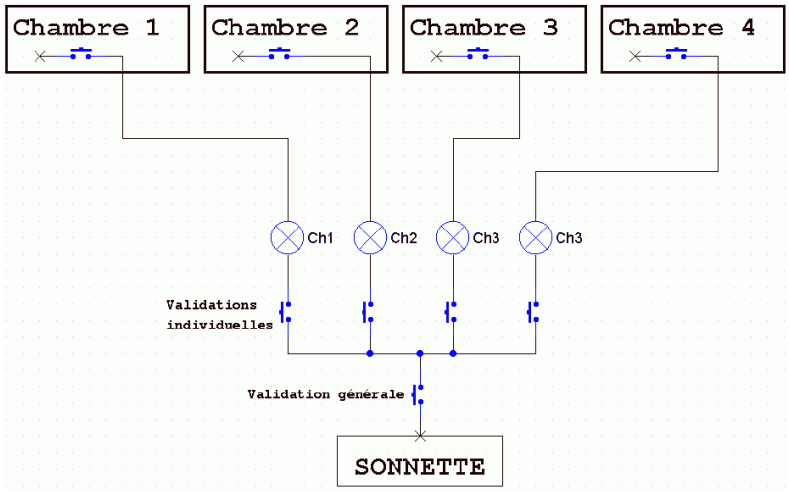
- TMR0 : Débordement du timer0 (tmr0). Une fois que le contenu du tmr0 passe de 0xff à 0x00, une interruption peut être générée.

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

- RB0/INT : Une interruption peut être générée lorsque, la pin RB0, encore appelée INTerrupt pin, étant configurée en entrée, le niveau qui est appliqué est modifié.

- PORTB : De la même manière, une interruption peut être générée lors du changement d'un niveau sur une des pins RB4 à RB7. Il n'est pas possible de limiter l'interruption à une seule de ces pins.

VII. LES INTERRUPTIONS SUR LES PICs



VII. LES INTERRUPTIONS SUR LES PICs

Le registre INTCON (INTerrupt CONtrol)

Ce registre se situe à l'adresse 0x0B, dans les 2 banques. Il est donc toujours accessible. C'est un registre de bits, donc, chaque bit a une fonction particulière.

b7 : GIE

Global Interrupt Enable bit. Il permet de valider ou d'invalider toutes les interruptions d'une seule fois. Ce bit correspond donc à notre interrupteur de validation générale.

b6 : EEIE

Eeprom write complete Interrupt Enable bit. Ce bit permet de valider l'interruption de fin d'écriture en eeprom (nous étudierons plus tard le mécanisme d'écriture eeprom).

VII. LES INTERRUPTIONS SUR LES PICs

b5 : T0IE: Tmr0 Interrupt Enable bit : Valide l'interruption générée par le débordement du timer0.

b4 : INTE: INTerrupt pin Enable bit : Valide l'interruption dans le cas d'une modification de niveau de la pin RB0.

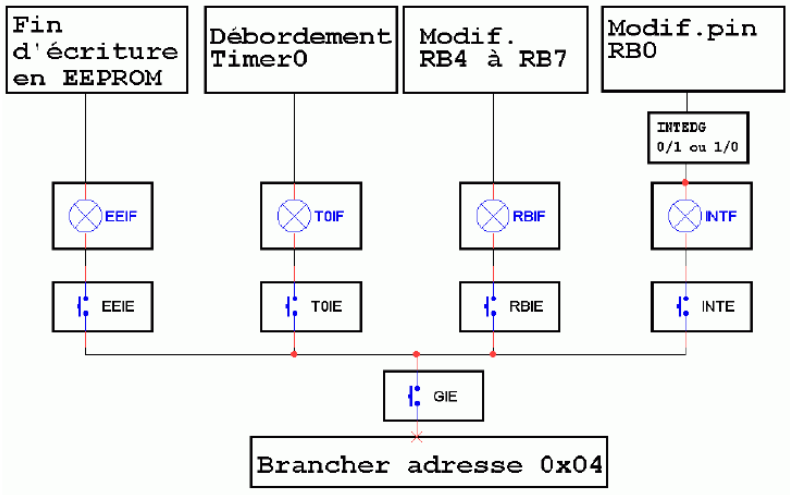
b3 : RBIE: RB port change Interrupt Enable bit : Valide les interruptions si on a changement de niveau sur une des entrées RB4 à RB7.

b2 : T0IF: Tmr0 Interrupt Flag bit. C'est un Flag, donc il signale. Ici c'est le débordement du timer0

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

b0 : RBIF: Port Interrupt Flag bit : signale qu'une des entrées RB4 à RB7 a été modifiée.

VII. LES INTERRUPTIONS SUR LES PICs



VIII. Le Timer

Le timer0 est un compteur, vous avez deux possibilités :
- En premier lieu, vous pouvez compter les impulsions reçues sur la pin RA4/T0KI. Nous dirons dans ce cas que nous sommes en mode compteur
- Vous pouvez aussi décider de compter les cycles d'horloge du PIC lui-même. Dans ce cas, comme l'horloge est fixe, nous compterons donc en réalité du temps. Donc, nous serons en mode « timer ».

La sélection d'un ou l'autre de ces deux modes de fonctionnement s'effectue par le bit 5 du registre OPTION : T0CS pour Tmr0 Clock Source select bit.
T0CS = 1 : Fonctionnement en mode compteur
T0CS = 0 : Fonctionnement en mode timer

VIII. Le Timer

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.

- o T0SE = 0 → incrémentation sur fronts montants
- o T0SE = 1 → incrémentation sur fronts descendants

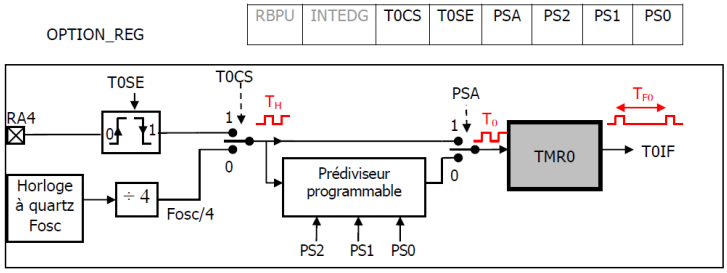
VIII. Le Timer

Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport DIV est fixés par les bits PS0, PS1 et PS2 du registre OPTION_REG

L'affectation ou non du prédiviseur se fait à l'aide du bit PSA du registre OPTION_REG

- o PSA = 0 → on utilise le prédiviseur
- o 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

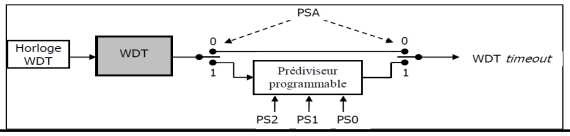


VIII. Le Timer

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 :

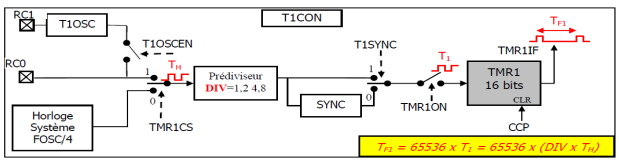
- Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur par un processus indésirable non contrôlé
- Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement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



VIII. Le Timer

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)



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

VIII. Le Timer

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

VIII. Le Timer

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.

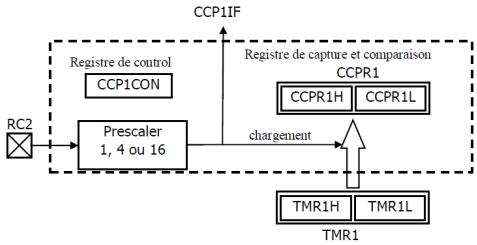
Ces modules ne fonctionnent pas si TMR1 est configuré en mode Compteur non synchronisé

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

VIII. Le Timer

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)

VIII. Le Timer

Le registre de configuration CCP1CON

—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

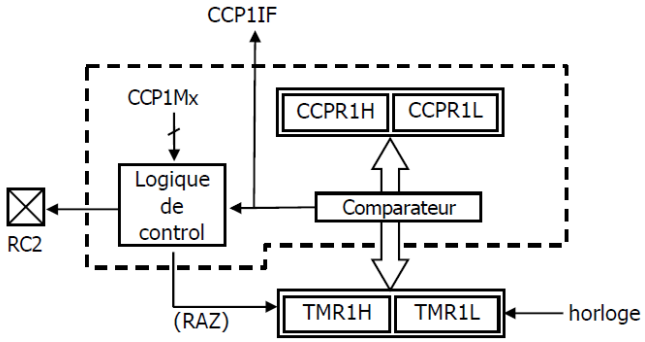
DC1B1, DC1B0 : utilisés en PWM mode Ce sont les 2 bits de poids faible des 10 bits MWM duty cycle. Les 8 autres bits sont dans le registre **CCPR1L CCP1M3 à CCP1M0 :** mode de fonctionnement du module CCP1 0000 : Module arrêté (reset module)

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 (drapeau CCP1IF + broche RC2 0 → 1)
1001 : Mode comparaison (drapeau CCP1IF + broche RC2 1→0)
1010 : Mode comparaison (drapeau CCP1IF seulement)
1011 : Mode comparaison (drapeau CCP1IF + RAZ TMR1)
10xx : PWM mode (modulation de largeur d'impulsion)

VIII. Le Timer

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



VIII. Le Timer

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.

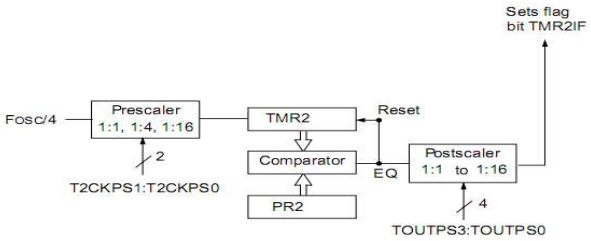
mode 1001 :
Au moment de l'égalité, le drapeau CCP1IF passe à 1 et le bit RC2 passe à 0. C'est à l'utilisateur de les remettre à leur état d'origine pour une prochaine utilisation. RC2 doit être configuré en sortie.

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

VIII. Le Timer

Le Timer TMR2



TMR2 est un timer 8 bits accessible en lecture écriture constitué de :
- un registre de control T2CON (bank0)
- un prédiviseur (1,4,16)
- un registre de période PR2 (bank1) accessible en lecture/écriture
- un comparateur,
- un postdiviseur (1 à 16)

IX. FORME GENERALE D'UN PROGRAMME

```
; Après chaque point virgule on peut mettre des commentaires

      list p=16f84,f=inhx8m          ; Type de PIC et format de fichier
      __config B'11111111110001'     ; Configuration du PIC
      #include "p16f84.inc"          ; Bibliothèque des
                                     ;instructions pour le PIC16F84


Son   equ   D'129'  ; Définition des constantes

Note  equ   H'0C'   ; Définition des Variables


      org   H'00'       ; Début du programme (non obligatoire)

     { les instructions du programme }

      end               ; Obligatoire
```

IX. FORME GENERALE D'UN PROGRAMME

➔ **Configuration du PIC**

__config B'11111111110001'

~ **Bit0** et **bit1**: fixe le type d'oscillateur:
11 => horloge avec un circuit RC
01 => horloge à quartz (4Mhz en général)
~ **Bit2**: Le chien de garde est activé lorsqu'il est à "1"

~ **Bit3**: La tempo à l'allumage du PIC est utilisée lorsqu'il est à "0" Cette tempo est utile pour permettre à l'oscillateur de se stabiliser.

~ **Bit4** à **Bit 13**: lorsqu'ils sont à "1", le programme n'est pas protégé, on peut toujours le récupérer du PIC. Dans le cas contraire, on ne peut pas lire le programme contenu dans le PIC, on peut juste l'exécuter.

IX. FORME GENERALE D'UN PROGRAMME

➔ **Structure d'un programme avec interruption**

```
list p=16f84,f=inhx8m              ; Type de PIC et format de fichier
    __config B'11111111110001'     ; Configuration du PIC
    #include "p16f84.inc"          ; Bibliotheque des
                                   ;instructions pour le PIC16F84
Son   equ   D'129'                 ; Définition des constantes

Note  equ   H'0C'                  ; Définition des Variables

    org   H'00'                    ; Début du programme (non obligatoire)
    goto  début

        org   H'04'                ; adresse d'interruption
        { programme d'interruption }
    retfie              ;retour à l'endroit ou le programme s'est interrompu
début
    { les instructions du programme }

    end
```

Les registres internes du PIC

Registre STATUS

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IRP	RP1	RP0	TO	$\overline{PD}$	$\overline{Z}$	DC	C

Registre INTCON

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

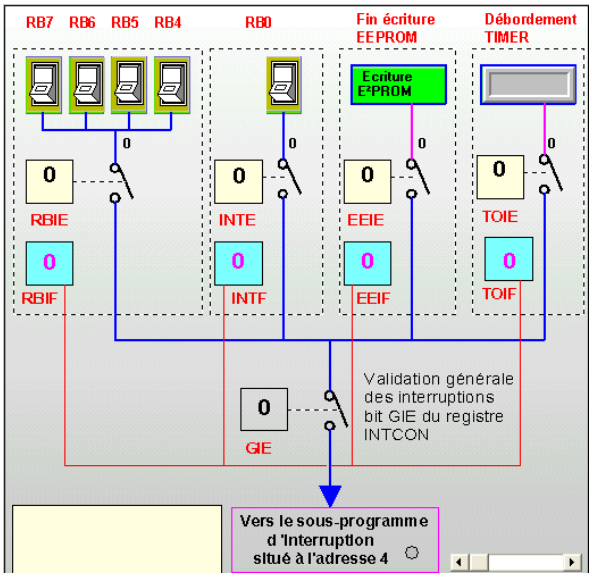
Registre OPTION

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
$\overline{RBPU}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Registre EECON1

it 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
			EEIF	WRERR	WREN	WR	RD

Les interruptions



X. EXEMPLES D'APPLICATIONS

Application avec un PIC : Gestion d'une interruption sur RB0

; Titre : Interruption sur RB0

Ce montage d'initiation à base de PIC permet de tester le déroulement d 'une interruption. Lorsque la broche RB0 passe de 0 à 1 (front montant) alors on génère une IT , on allume une led (RB7) et après une temporisation on éteint celle-ci.

Une prochaine action sur RB0 redéclenche l ' IT

X. EXEMPLES D'APPLICATIONS

```
        list p=16f887
        #include p16f887.inc
        __config H'3FF9'

;**** Le programme principal commence à l ' étiquette init ******
        ORG 0
        goto init
;**** Le programme d' interruption se déclenche lorsque l' entrée RB0 passe de 0 à 1 ***
        ORG 4
;********** Programme d 'interruption **********************
        bsf PORTB,7          ; on allume la led connectée sur rb7
;------------  temporisation  ---------------
tempo2
        MOVLW 0xFF           ; on met ff dans le registre W
        MOVWF retard1        ; on met W dans le registre retard1
        MOVWF retard2        ; on met W dans le registre retard2
        MOVLW 0x07           ; on met 7 dans le registre W
        MOVWF retard3        ; on met W dans le registre retard3
```

X. EXEMPLES D'APPLICATIONS

```
attente2

        DECFSZ retard1,F      ; on décrémente retard1 et on saute la prochaine instruction si
        GOTO  attente2        ; le registre retard1 = 0 sinon retour à tempo

        movlw 0xFF            ; on recharge retard1
        movwf retard1

        DECFSZ retard2,F      ; on décrémente retard2 et on saute la prochaine instruction si
        GOTO  attente2        ; le registre retard2 = 0 sinon retour à tempo

        movlw 0xFF            ; on recharge retard2
        movwf retard2

        DECFSZ retard3,F      ; on décrémente retard3 et on saute la prochaine instruction si
        GOTO  attente2        ; le registre retard3 = 0 sinon retour à tempo

        bcf PORTB,7           ; on éteint la led connectée sur rb7
        bcf INTCON,INTF         ; on remet à 0 le bit du registre d' IT qui est passé à 1

        RETFIE              ; retour d 'interruption
```

X. EXEMPLES D'APPLICATIONS

```
;************* Programme d' INIT ***************************
init
retard1     EQU      0x0C          ; le registre temporaire retard1 se trouve à l' adresse 0C
retard2     EQU      0x0F          ; le registre temporaire retard2 se trouve à l' adresse 0F
retard3     EQU      0x0E          ; le registre temporaire retard3 se trouve à l' adresse 0F
        bsf STATUS,5               ; on met à 1 le 5eme bit du registre status pour accéder
                                   ; à la 2eme page mémoire ( pour configurer trisa et trisb
                                   ; -> broches en entrée ou en sortie )
        MOVLW B'00000001'          ; rb0, en entrée ( rb0 sera la broche utilisée
                                   ; pour l ' interruption )
        MOVWF TRISB
        bcf STATUS,5               ; on remet à 0 le 5eme bit du registre status pour accéder
                                   ; à la 1ere page mémoire
        bsf OPTION_REG,INTEDG      ; Le passage de 0 à 1 sur RB0 provoque une IT
                                   ; sur un  front montant ( choix du front sur RB0 )
        bsf INTCON,INTE            ; autorise l ' IT sur RB0
        bsf INTCON,GIE             ; autorise les Interruptions
        clrf PORTB
;************ Programme principal en rebouclage **********************
debut
        sleep                  ; mise en sommeil du PIC conso : 3.2 mA, attente impulsion sur RB0
        GOTO debut
;*********** Fin du programme ***********
        end
```

X. EXEMPLES D'APPLICATIONS

Ecrire un programme qui fait clignoter une LED en utilisant les interruptions générées par le timer interne (TMR0)

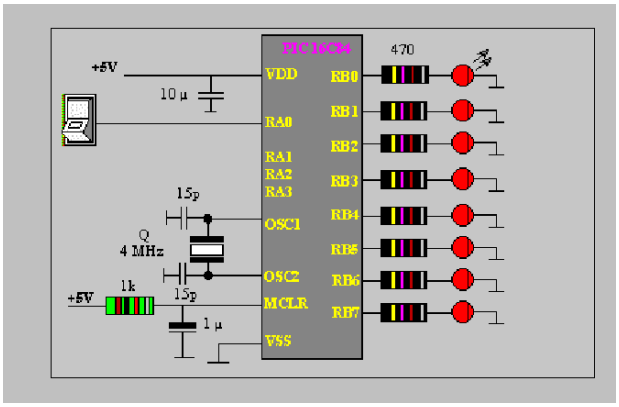
```
SAVE_W EQU 0C              ;Déclaration de
SAVE_STAT EQU 0D           ;deux variables.
; VECTEURS
ORG 00                     ;Vecteur de Reset.
GOTO START
ORG 04                     ;Vecteur d'interruption.
GOTO INT_VECT

 START ORG 05
                           ; INITIALISATIONS
 BSF STATUS,RP0            ;On passe en Page 1.
 MOVLW b'00000000'         ;Port B en sortie.
 MOVWF TRISB
```

X. EXEMPLES D'APPLICATIONS

```
MOVLW b'00000111'          ;On configure OPTION.
MOVWF OPTION_REG           ;Le pré diviseur divise par 255.
BCF STATUS,RP0             ;On revient en Page 0.
CLRF TMR0                  ;Timer à zéro.
CLRF PORTB                 ;Toutes LED éteintes.
MOVLW b'10100000'          ;On configure INTCON.
MOVWF INTCON               ; - GIE (bit 7) à 1 - T0IE (bit 5) à 1
                           ; - tous les autres bits à zéro


LOOP GOTO LOOP             ;Boucle introduite juste pour
                           ; occuper le processeur, car le
                           ; but du programme est
                           ; d'attendre l'apparition du
                           ; signal d'interruption.
```

X. EXEMPLES D'APPLICATIONS

réaliser un chennillard sur les broches RB0 à RB5 d' un PIC
; le quartz est de 4 Mhz , on effectue une tempo environ égale à 0.2 seconde
; un bouton marche sur le port A permet de lancer l' application



X. EXEMPLES D'APPLICATIONS

```
;------------  Définition des registres  ----------------

retard1   EQU    0x0C    ; le registre temporaire retard1 se trouve à l' adresse 0C
retard2   EQU    0x0F    ; le registre temporaire retard2 se trouve à l' adresse 0F
memo      EQU    0x10    ; le registre temporaire memo se trouve à l' adresse 10

;------------  Init des ports A et B  ----------------

      ORG 0

      bsf STATUS,5          ; on met à 1 le 5eme bit du registre status pour accéder
                            ; à la 2eme page mémoire ( pour trisa et trisb )

      MOVLW 0x00            ; on met 00 dans le registre W
      MOVWF TRISB           ; on met 00 dans le port B il est programmé en sortie

      MOVLW 0x1F            ; on met 1F dans le registre W
      MOVWF TRISA           ; on met 1F dans le port A il est programmé en entrée

      bcf STATUS,5          ; on remet à 0 le 5eme bit du registre status pour accéder
                            ; à la 1eme page mémoire
```

X. EXEMPLES D'APPLICATIONS

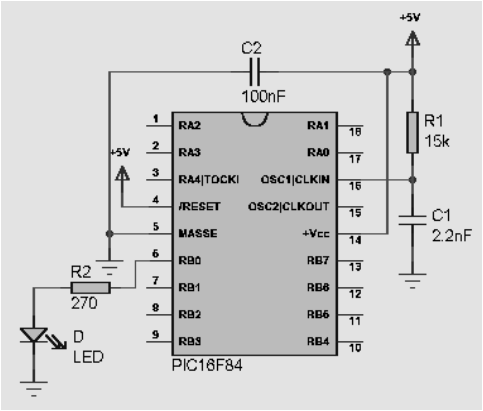
```
;------------  Init des leds  et registre ----------------

        CLRF  PORTB             ; on met 0 sur le port B ( leds )
        MOVLW  01               ; on met 01 dans le registre W
        MOVWF  memo             ; on met W dans le registre memo

;--------------------  Programme principal  ----------------------
debut
        btfss PORTA,inter0          ; interrupteur 0 ( marche ) appuyé ? si oui on continu sinon
                                    ;va à debut
        goto debut

        movf memo,W             ; on met memo dans W
        movwf PORTB             ; on met W sur le port B ( leds )


        CALL  tempo               ; on appel la temporisation

        RLF memo, F               ; rotation à gauche du registre memo
        GOTO  debut               ; retour au début du programme
```

X. EXEMPLES D'APPLICATIONS

```
;------------  Programme de temporisation longue  ----------------

tempo
        MOVLW 0xff          ; on met ff dans le registre W
        MOVWF retard1        ; on met W dans le registre retard1
        MOVWF retard2        ; on met W dans le registre retard2

attente

        DECFSZ retard1,F      ; on décrémente retard1 et on saute la prochaine instruction si
        GOTO  attente         ; le registre retard1 = 0 sinon retour à tempo

        movlw 0xFF        ; on recharge retard1
        movwf retard1

        DECFSZ retard2,F      ; on décrémente retard2 et on saute la prochaine instruction si
        GOTO  attente         ; le registre retard2 = 0 sinon retour à tempo


        RETURN              ; retour au programme principal après l 'instruction CALL

        END
```

X. EXEMPLES D'APPLICATIONS

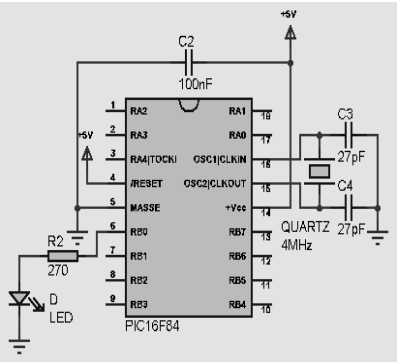
Exemple1: clignotement d'une LED à l'aide du chien de garde.




```
__config  B'11111111110111'
#include "p16f887.inc"
 bsf     STATUS,RP0
movlw   B'00001101'
movwf   OPTION_REG
movlw   B'11111110'
movwf   TRISB
bcf     STATUS,RP0
Boucle    sleep
comf    PORTB,1
goto    Boucle
end
```

X. EXEMPLES D'APPLICATIONS

Exemple2: clignotement d'une LED en créant des retards dans le PIC.




```
__config  B'11111111110001'
      #include "p16f887.inc"
temp1     equ    H'0C'
temp2     equ    H'0D'
      bsf     STATUS,RP0
      movlw   B'11111110'
      movwf   TRISB
      bcf     STATUS,RP0
      clrf    temp1
Debut     movlw   D'244'
      movwf   temp2
Boucle      nop
      decfsz  temp1,1
      goto    Boucle
      decfsz  temp2,1
      goto    Boucle
      comf    PORTB,1
      goto    Debut
      end
```

X. EXEMPLES D'APPLICATIONS

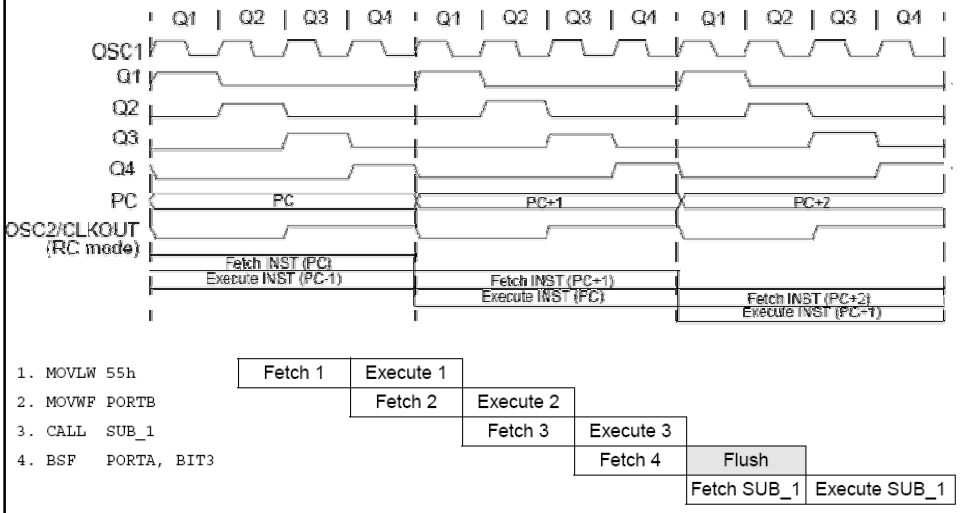
Exemple3: clignotement d'une LED à l'aide des interruptions du TIMER 0.

```
__config  B'11111111110001'
      #include "p16f887.inc"
temps      equ H'0C'
      org H'00'
      goto Debut
      org H'04'
      movlw D'012'
      movwf TMR0
      bcf INTCON,T0IF
      decfsz temps,1
      retfie
      comf PORTB,1
      movlw D'008'
      movwf temps
      retfie
Debut      bsf STATUS,RP0
      movlw B'10000111'
      movwf OPTION_REG
      bcf TRISB,0
      bcf STATUS,RP0
      movlw D'008'
      movwf temps
      movlw B'10100000'
      movwf INTCON
Boucle     goto Boucle
      end
```

Cycle instruction

- 1 cycle instruction (Tcy) est décomposé en 4 étapes Q1 – Q4
  - Q1: Décodage d'instruction (ou nop)
  - Q2: Lecture (ou nop)
  - Q3: Calcul
  - Q4: Ecriture (ou nop)
- période Q = période oscillateur (Tosc)
  ➔fcy = fosc/4 !!!
- Une instruction s'exécute entièrement en 2 cycles instructions
- 1ier Tcy : fetch (gestion de PC, chargement instruction dans Instruction Reg)
- 2ième Tcy : décodage et exécution
- « Pipelining » : traitement en parallèle du fetch et du « décodage-exécution »!

Cycle instruction



1. MOVLW 55h	Fetch 1	Execute 1			
2. MOVWF PORTB		Fetch 2	Execute 2		
3. CALL SUB_1			Fetch 3	Execute 3	
4. BSF PORTA, BIT3				Fetch 4	Flush
					Fetch SUB_1	Execute SUB_1

Accès aux mémoires

- RAM en lecture / écriture
  → Pas de problèmes
- EEPROM en lecture / écriture
- ROM en lecture / écriture ?

Accès à l'EEPROM et à la ROM

- Accès via des SFR
- EEPROM et ROM : même méthode
- 6 registres utilisés
  - EEDATAH:EEDATA → donnée
  - EEADRH:EEADR → adresse à lire/écrire
  - EECON1 → registre de contrôle et de paramétrage des accès
  - EECON2 → registre de contrôle en écriture
    (séquences de valeur pour l'écriture)

EECON1 (adr : 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit
1 = Accesses program memory
0 = Accesses data memory
Reads '0' after a POR, this bit cannot be changed while a write operation is in progress.

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{\text{MCLR}}$ or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read; RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Lire en EEPROM

1. Ecrire dans EEADR l'adresse à lire
2. Choisir un accès à l'EEPROM
   0 → EECON1<EEPGD>
3. Démarrer la lecture
   1 → EECON1<RD>
4. Lire la valeur EEDATA

Code associé :

```
BSF     STATUS,RP1
BCF     STATUS,RP0
MOVF    DATA_EE_ADDR,W
MOVWF   EEADR
BSF     STATUS,RP0
BCF     EECON1,EEPGD

BSF     EECON1,RD
BCF     STATUS,RP0
MOVF    EEDATA,W
```

Ecrire en EEPROM (sans IRQ)

1. Vérifier qu'une écriture ne soit pas en cours
2. Choisir l'adresse EEPROM et la valeur à écrire
3. Choisir un accès à l'EEPROM
4. Autoriser l'écriture: 1→ EECON1<WREN>
5. Executer la séquence :
   55h → EECON2
   AAh → EECON2
   1 → EECON1<WR>
6. Interdire l'écriture: 0→ EECON1<WREN>

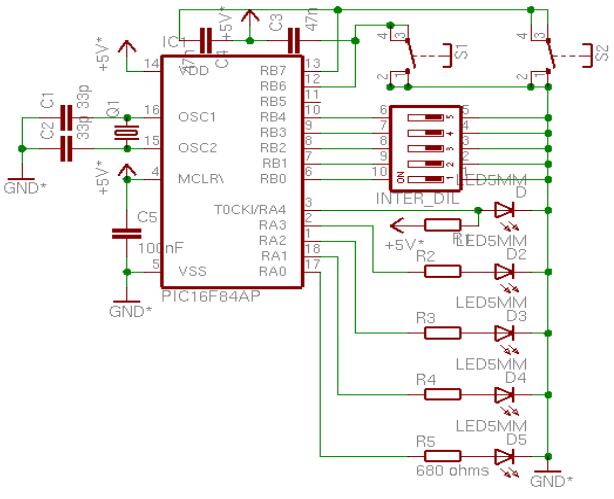
Ecrire en EEPROM, code

```
BSF     STATUS,RP1      ;
BSF     STATUS,RP0
BTFSC   EECON1,WR       ;Wait for write
GOTO    $-1             ;to complete
BCF     STATUS,  RP0    ;Bank 2
MOVF    DATA_EE_ADDR,W  ;Data Memory
MOVWF   EEADR           ;Address to write
MOVF    DATA_EE_DATA,W  ;Data Memory Value
MOVWF   EEDATA          ;to write
BSF     STATUS,RP0      ;Bank 3
BCF     EECON1,EEPGD    ;Point to DATA
                        ;memory
BSF     EECON1,WREN     ;Enable writes

BCF     INTCON,GIE      ;Disable INTs.
MOVLW   55h             ;
MOVWF   EECON2          ;Write 55h
MOVLW   AAh             ;
MOVWF   EECON2          ;Write AAh
BSF     EECON1,WR       ;Set WR bit to
                        ;begin write
BSF     INTCON,GIE      ;Enable INTs.
BCF     EECON1,WREN     ;Disable writes
```

Required Sequence

X. EXEMPLES D'APPLICATIONS

Exemple3: Lecture et écriture dans la mémoire EEPROM du PIC.



X. EXEMPLES D'APPLICATIONS

```
B'11111111110001'
include "p16f887.inc"
bsf STATUS,RP0
movlw B'11100000'
movwf TRISA
bcf OPTION_REG,7
bcf STATUS,RP0
clrf PORTA
clrf EEADR
Boucle
btfss PORTB,6
call Enregistre
btfss PORTB,7
call Lit
goto Boucle
```

```
Enregistre
btfss PORTB,6
goto Enregistre
comf PORTB,0
movwf EEDATA
movwf PORTA
call Ecriture
incf EEADR,1
return
Lit
btfss PORTB,7
goto Lit
call Lecture
movf EEDATA,0
movwf PORTA
incf EEADR,1
return
```

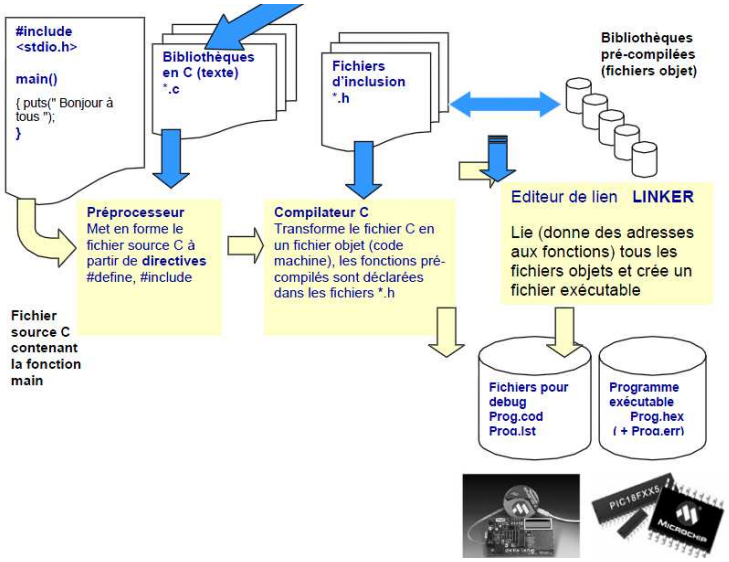
X. EXEMPLES D'APPLICATIONS

```
;Écriture EEPROM  Ecriture
bsf STATUS,RP0
clrf EECON1
bsf EECON1,WREN
movlw H'55'
movwf EECON2
movlw H'AA'
movwf EECON2
bsf EECON1,WR EcritureFin
btfsc EECON1,WR
goto EcritureFin
bcf STATUS,RP0
Return
```

```
; Lecture EEPROM
Lecture
bsf STATUS,RP0
bsf EECON1,RD
bcf STATUS,RP0
return
;Écriture des données dans la mémoire EEPROM du ;PIC à l'aide du programmateur
org H'2100'
de B'00000001',B'00000010',B'00000100',B'00001000' de B'00010000',B'00010000',B'00001000',B'00000100' de B'00000010',B'00000001',B'00000000'
end
```

PROGRAMMATION DES PICs EN C

SCHEMA GENERAL DU PROCESSUS DE COMPILATION



Directives du pr pré-processeur

#include

Sert à inclure un fichier contenant du code source (.c ou .h) dans un autre fichier.

#include<Nomfichier> : recherche du fichier dans :

Les répertoires mentionnéss à l'aide de l'option de compilation / Idirectory

Les répertoires définis à l'aide de la variable d'environnement INCLUDE

#include"Nomfichier": recherche du fichier dans :

Idem cas précédent + Le répertoire courant

Exemples :

- PORTB=0xA4 ; ou a=PORTB ;
- PORTBbits.RB0=0 ; ou PORTBbits.RB0=1 ;
- On utilise LATBbits.LATB0 pour accéder au latch B0.
- If (PORTAbits.RA4) … ; else …. ; L'expression sera vraie si PORTA4 est non nul, il est donc inutile d'écrire (PORTAbits.RA4==1)

p16f887.h

```
extern volatile near unsigned char PORTA;
extern volatile near union  {
   struct {
      unsigned RA0:1;
      unsigned RA1:1;
      unsigned RA2:1;
      unsigned RA3:1;
      unsigned RA4:1;
      unsigned RA5:1;
      unsigned RA6:1;
   } ;
   struct {
      unsigned AN0:1;
      unsigned AN1:1;
      unsigned AN2:1;
      unsigned AN3:1;
      unsigned :1;
      unsigned AN4:1;
      unsigned OSC2:1;
   } ;
   struct {
      unsigned :2;
      unsigned VREFM:1;
      unsigned VREFP:1;
      unsigned T0CKI:1;
      unsigned SS:1;
      unsigned CLK0:1;
   } ;
   struct {
      unsigned :5;
      unsigned LVDIN:1;
   } ;
} PORTAbits ;
```

→ Le port A est un octet (unsigned char) défini dans un fichier externe (extern) dont la valeur peut être écrasée entre 2 appels (volatile).

→ La deuxième déclaration précise que PORTAbits est une union de structures **anonymes** de bits adressables.
Du fait que chaque bit d'un registre de fonction peut avoir plusieurs affectations, il y peut y avoir plusieurs définitions de structures à l'intérieur de l'union pour un même registre.

Dans le cas présent les bits du port A sont définis comme :

1ère structure :
port d'E/S parallèle (7 bits ; RA0 à RA6)
2ème structure :
port d'entrées analogiques (5 entrées AN0 à AN4) + entrée OSC2.
3ème structure :
Des entrées de tension de référence du CAN, entrée horloge externe du timer0 (T0CKI), entrée de sélection du port série synchrone (SS), sortie du timer0 (CLK0).
4ème structure :
entrée low voltage detect (LVDIN)

Le contenu du registre ADCON1 déterminera l'affectation d'un bit (cf DS39564B page 182).

L'accès à un bit du portA se fait de la façon suivante :

Nom_union.nom_bit

Exemple :
PORTAbits.RA0 = 1 ; // mise à l'état haut de RA0

Quelques instructions spécifiques au PIC

Définitions des broches: ex: "#define LED pin_c7" La variable LED correspond au bit 7 du PORTC.

Mise à zéro ou à un d'une sortie: ex: "output_high(LED);" ou "output_low(LED);"

La broche correspondant à LED est soit mise à un, soit mise à zéro.

Lecture de l'état d'une entrée: ex: "input(SW1)" Cette instruction renvoie 0 ou 1 correspondant à l'état logique de la broche nommée SW1.

Il peut être rapidement nécessaire de connaître les instructions suivantes:

Temporisation: Il existe les instructions de temporisation en millisecondes ("delay_ms()") ou en microsecondes ("delay_us()")ex: delay_us(25); = temporisation de 25µs.

Quelques instructions spécifiques au PIC

Ecriture de 8 bits sur un port: ex "output_a (value)" Il y a écriture de la donnée 'value' sur le port A, pour les autres ports il faut modifier la lettre du port (ex: output_b pour le port B).

Lecture des 8 bits d'un port: ex: "value = input_a()" La variable 'value' prend la valeur correspondante au code formé par les 8 bits du port A, comme précédemment, if faut changer le lettre pour les autres ports.

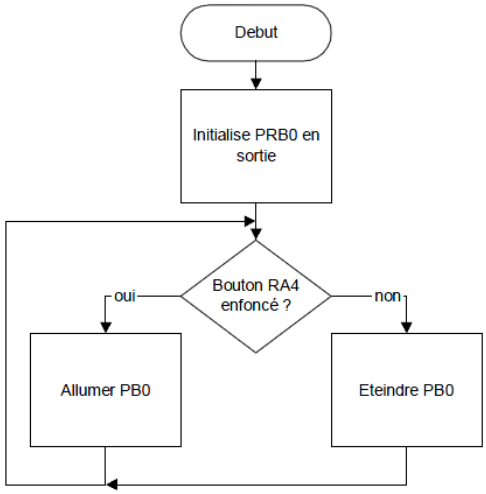
Validation ou dévalidation des résistances de rappel du portB: " port_b_pullups(FALSE ou TRUE);"

Ecriture dans les registres de direction TRIS: "set_tris_a (value)" Ecriture de 'value' dans TRISA. Pour les autres ports, changer la lettre.

Quelques instructions spécifiques au PIC

```
#include <pic.h>
#define _XTAL_FREQ  20000000
__CONFIG(HS & WDTDIS & LVPDIS);
void main(){
TRISB = 0; // Set PORTB output mode
while(1){
RB0 = 1; // LED ON
__delay_us(1); //
RB2=0; // LED OFF
__delay_us(1);
}
}
```

Gestion des ports parallèles



Gestion des ports parallèles

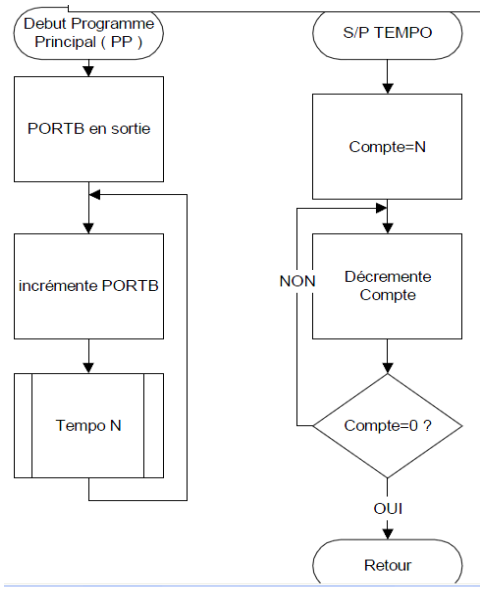
```
/* La LED sur PB0 s'éteint si S2 (PA4) est enfoncé*/
#include <p18f452.h>
void main(void)
{ TRISA=0xFF; // PORTA en entrée
TRISB = 0; /* PB en sortie */
while(1) // une boucle infinie
{
if (PORTA & 0x10) PORTB=1;
else PORTB=0;
}
}
```

Modifier ce programme afin d'incrémenter PRB à chaque pression sur RA4.
(pour tester RA4 : while(PORTAbits.RA4)

Remarques :

- seule la LED sur PB0 devant être modifiée, on aurait pu écrire : PORTB=PORTB|0b00000001; pour mettre PB0 à 1 et PORTB=PORTB&0b11111110; pour mettre PB0 à 0.
- Très souvent les masques sont utilisés en C pour les tests ou les positionnements de bit,

Création d'une fonction



Création d'une fonction

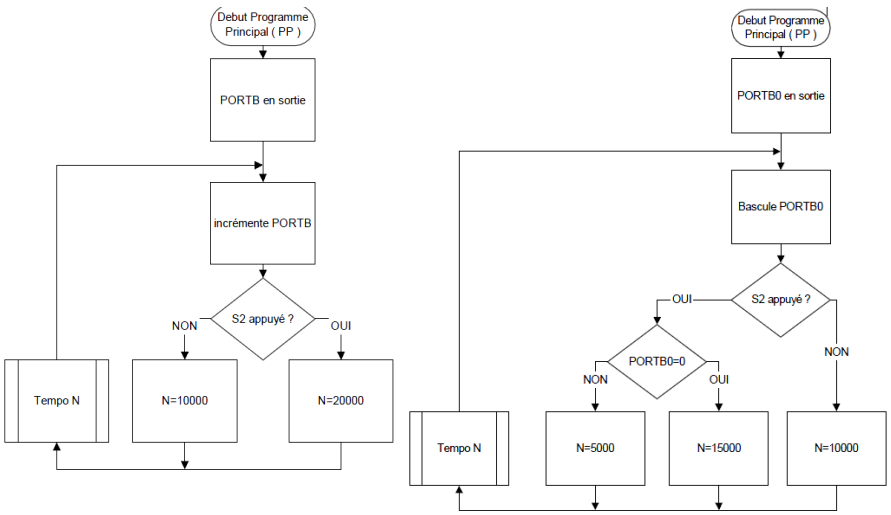
```
#include <p16f887.h>
#define duree 10000
void tempo(unsigned int count);
void main(void)
{
PORTB = 0x00;
TRISB = 0x00;
while(1) {
PORTB++;
tempo(duree);
}
}
void tempo(unsigned int compte)
{
while(compte--);
}
```

Remarque : Si une fonction est écrite avant son appel le prototype devient inutile

- Modifier le programme de manière à modifier la tempo (passer de 10000 à 20000) si S2 est appuyé.

- Réaliser un programme faisant clignoter RB0 avec une période proche de 1s et un rapport cyclique ¼ si S2 est appuyé et ½ sinon

Création d'une fonction



Décalages

Utilisation des opérateurs de décalage gauche et droite, ces derniers permettent également des multiplications et divisions par deux très rapides. (Filtre numérique par exemple)

```
#include <p18f452.h>
void wait(int cnt)
{
for (;cnt>0; cnt--);
}
void main(void)
{
int x;
char c=0;
TRISB = 0;
PORTB=0b00000001;
while(1)
{
if (PORTB==8) c++;
if (PORTB==1) c--;
if (!c) PORTB>>=1;
else PORTB<<=1;
if (PORTA&0x10) x= 20000;
else x=5000;
wait(x);
}
}
```

Equivalence avec le langage C

Charger une valeur littérale dans un registre

```
movlw B'10001100' ; W = B'10001100'
movwf REGISTRE ; (REGISTRE) = B'10001100' = 0x8C = D'140'
```

Equivalence en langage C :

```
REGISTRE = 0x8C ;
```

Charger un registre avec le contenu d'un autre registre

```
movf REGISTRE1 , W ; W = (REGISTRE1)
movwf REGISTRE2 ; (REGISTRE2) = (REGISTRE1)
```

Equivalence en langage C :

```
REGISTRE2 = REGISTRE1 ;
```

Equivalence avec le langage C

Echanger le contenu de deux registres : (REGISTRE1) < - > (REGISTRE2)

Il faut utiliser une variable intermédiaire :
REGISTRE_TEMP (registre d'usage général)

```
movf REGISTRE1 , W  ; W = (REGISTRE1)
movwf REGISTRE_TEMP ; (REGISTRE_TEMP) = (REGISTRE1)
movf REGISTRE2, W   ; W = (REGISTRE2)
movwf REGISTRE1     ; (REGISTRE1) = (REGISTRE2)
movf REGISTRE_TEMP, W ; W = (REGISTRE_TEMP)
movwf REGISTRE2     ; (REGISTRE2) = (REGISTRE_TEMP)
```

Equivalence en langage C :

```
REGISTRE_TEMP = REGISTRE1 ;
REGISTRE1 = REGISTRE2 ;
REGISTRE2 = REGISTRE_TEMP ;
```

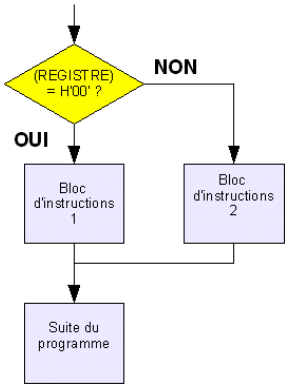
Equivalence avec le langage C

Tests de comparaison

Tests d'égalité

Le contenu du registre est-il nul ?

(REGISTRE) = 0x00 ?




```
movf REGISTRE , f ; (REGISTRE) = (REGISTRE)
btfss STATUS , Z ; test du bit Z
goto non ; Z = 0 c-à-d(REGISTRE) != 0x00
{ bloc d'instructions 1 }
goto suite
non
{ bloc d'instructions 2 }
suite
{ suite du programme }
```

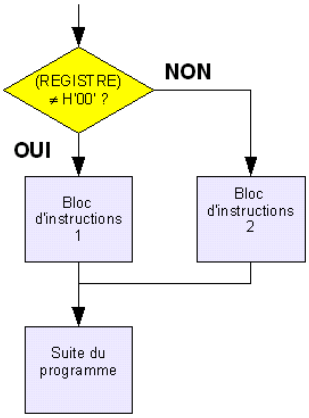
Equivalence en langage C :

```
if (REGISTRE = = 0x00)
{
// bloc d'instructions 1
}
else
{
// bloc d'instructions 2
}
```

Equivalence avec le langage C

Le contenu du registre est-il différent de zéro ?

(REGISTRE) != 0x00 ?




```
movf REGISTRE , f ; (REGISTRE) = (REGISTRE)
btfsc STATUS , Z ; test du bit Z
goto non ; Z = 1 c-à-d (REGISTRE) = 0x00
{ bloc d'instructions 1 }
goto suite
non
{ bloc d'instructions 2 }
suite
{ suite du programme }
```

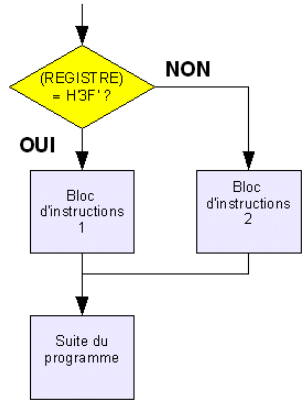
Equivalence en langage C :

```
if (REGISTRE != 0x00)
{
// bloc d'instructions 1
}
else
{
// bloc d'instructions 2
}
```

Equivalence avec le langage C

Le contenu du registre est-il égal à une certaine valeur ?

(REGISTRE) = 0x3F ?



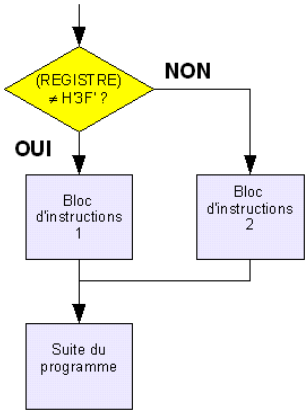

```
if (REGISTRE = = 0x3F)
{
// bloc d'instructions 1
}
else
{
// bloc d'instructions 2
}
```

```
movlw 0x3F ; W = 0x3F
subwf REGISTRE, W ; W = (REGISTRE) - 0x3F
btfss STATUS, Z ; test du bit Z
goto non ; Z = 0 c-à-d (REGISTRE) != 0x3F
{ bloc d'instructions 1 }
goto suite
non
{ bloc d'instructions 2 }
suite
{ suite du programme }
```

Equivalence avec le langage C

Le contenu du registre est-il différent d'une certaine valeur ?

(REGISTRE) != 0x3F ?



En langage C :

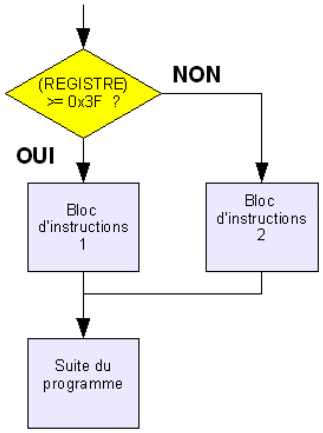
```
if (REGISTRE != 0x3F)
{
// bloc d'instructions 1
}
else
{
// bloc d'instructions 2
}
```

```
movlw 0x3F ; W = 0x3F
subwf REGISTRE, W ; W = (REGISTRE) - 0x3F
btfsc STATUS, Z ; test du bit Z
goto non ; Z = 1 c-à-d (REGISTRE) = 0x3F
{ bloc d'instructions 1 }
goto suite
non
{ bloc d'instructions 2 }
suite
{ suite du programme }
```

Equivalence avec le langage C

Le contenu du registre est-il strictement supérieur à une certaine valeur ?

(REGISTRE) >= 0x3F ?



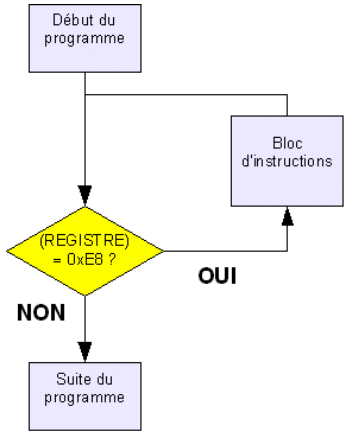
En langage C :

```
if (REGISTRE >= 0x3F)
{
// bloc d'instructions 1
}
else
{
// bloc d'instructions 2
}
```

```
movlw 0x3F ; W = 0x3F
subwf REGISTRE, W ; W = (REGISTRE) - 0x3F
btfss STATUS, C ; test du bit C (Carry)
goto non ; C = 0 c-à-d (REGISTRE) < 0x3F
{ bloc d'instructions 1 }
goto suite
non
{ bloc d'instructions 2 }
suite
{ suite du programme }
```

Equivalence avec le langage C

Boucle While




```
debut
movlw 0xE8 ; W = 0xE8
subwf REGISTRE, W ; W = (REGISTRE) -
0xE8
btfss STATUS , Z
goto suite
{ bloc d'instructions }
goto debut
suite
{ suite du programme }
```

Equivalence en langage C :

```
while (REGISTRE = = 0xE8)
{
// bloc d'instructions
}
```

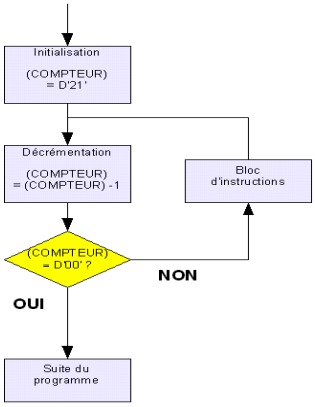
Equivalence avec le langage C

Boucle FOR

Une variable (1 octet) sert de compteur.

Exemple :

Pour exécuter le bloc d'instructions 20 fois, la valeur initiale du compteur doit être 21 :




```
movlw D'21' ; W = D'21'
movwf COMPTEUR ; (COMPTEUR) = 20 boucles
debut
decfsz COMPTEUR , f;(COMPTEUR) = (COMPTEUR) - 1
goto boucle ; (COMPTEUR) != 0
goto suite ; (COMPTEUR) = 0
boucle
{ bloc d'instructions }
goto debut
suite
{ suite du programme }
```

Equivalence en langage C

```
for (COMPTEUR = 20 ; COMPTEUR > 0 ; COMPTEUR--)
{
// bloc d'instructions
}
```

Labs en C

Ecrire un programme qui fait clignoter une LED sur le port C du pic pendant 0.5 s

```
#include <pic.h> // Include header file for MCU
#define _XTAL_FREQ    20000000 // Define Frequency 20.0 MHz for function
__delay_ms
__CONFIG(HS & WDTDIS & LVPDIS);    // Config. High speed clock,Disable
watchdog and Disable LVP
void Delay_ms(unsigned int tick)
{
     while(tick--)    // Loop couter delay time
     {
       __delay_ms(1);  // Delay 1 ms
     }
}
void main()
{
    TRISC = 0x00;         // Set PORTC output mode
    while(1) // Infinite loop
    {
        PORTC = 0x00;     // LED at PORTC ON
        Delay_ms(500);    // Delay 0.5 sec
        PORTC = 0xFF;     // LED at PORTC OFF
        Delay_ms(500);    // Delay 0.5 sec
    }
}
```

Labs en C

```
#include <pic.h> // Include header file for MCU
#define _XTAL_FREQ    20000000 // Define Frequency 20.0 MHz for function __delay_ms
__CONFIG(HS & WDTDIS & LVPDIS);    // Config. High speed clock,Disable watchdog
and Disable LVP
void Delay_ms(unsigned int tick)
{
     while(tick--)    // Loop couter delay time
     {
      __delay_ms(1);  // Delay 1 ms
     }
}
void main()
{
    TRISC = 0x00;         // Set PORTC output mode
    while(1) // Infinite loop
    {
        PORTC = ~PORTC;   // LED at PORTC tooggle
        Delay_ms(500);    // Delay 0.5 sec
    }
}
```

Labs en C

Ecrire un programme qui fait clignoter une LED sur le RD0 du port D du pic pendant 1 s

Labs en C

```
#include <pic.h> // Include header file for MCU
__CONFIG(HS & WDTDIS & LVPDIS);   // Config. High speed clock,Disable watchdog
and Disable LVP
unsigned int ms=0;    // Keep Counter every 1 ms
void main()
{
    TRISD0 = 0;  // Set RD0 output mode
    // Timer 0 Prescaler 1:128
    PS0 = 0;
    PS1 = 1;
    PS2 = 1;
    PSA = 0;// Prescaler use for Timer 0
    TMR0 = 216;  // Initial value for Timer 0
    T0CS = 0;    // Use internal clock source
    while(1) // Infinite loop
    {
        if(T0IF==1)  // Timer 0 overflow?
        {
            TMR0 = 216;      // Reload value for Timer 0
            T0IF = 0;        // Clear Timer 0 overflow flag
            ms++;    // Increase when 1 ms
            if(ms>=1000) // Up to 1 sec.?
            {
                ms=0;    // Clear counter for next time
                RD0 =~RD0;   // Toggle LED at RD0
            }
        }
```

Labs en C

Ecrire un programme qui fait incrémente le port C si RB0 =1 et décrémente C si RA4 = 1 et mise à 0 le port C si RE1 = 0. le passage entre les différentes états prend 0.5s

Labs en C

```
#include <pic.h> // Include header file for MCU
#define _XTAL_FREQ    20000000 // Define Frequency 20.0 MHz for function
__delay_ms
__CONFIG(HS & WDTDIS & LVPDIS);    // Config. High speed clock,Disable watchdog
and Disable LVP
void Delay_ms(unsigned int tick)
{
     while(tick--)    // Loop counter delay time
     {
      __delay_ms(1);  // Delay 1 ms
     }
}
```

Labs en C

```
void main()
{
     TRISC = 0;        // Set PORTC output mode
     PORTC = 0;        // Clear port
     ANS6 = 0;         // Set RE1 as digital port
     ANS12 = 0;        // Set RB0 as digital port
     TRISB0 = 1;       // Set RB0 input mode
     TRISA4 = 1;       // Set RA4 input mode
     TRISE1 = 1;       // Set RE1 input mode
     while(1) // Infinite loop
     {
          if(RB0==0)   // Switch at RB0 press?
          {
               PORTC++; // Increase data of PORTC 1 time
               Delay_ms(200);        // Delay 0.5 sec
          }
          if(RA4==0)   // Switch at RA4 press?
          {
               PORTC--; // Decrease data of PORTC 1 time
               Delay_ms(200);        // Delay 0.5 sec
          }
          if(RE1==0)   // Switch at RE1 press?
          {
               PORTC = 0;   // Clear data of PORTC
               Delay_ms(200);        // Delay 0.5 sec
          }
```

Labs en C

Ecrire un programme qui fait basculer la valeur du port C, une fois une interruption se déclenche sur RB0 du port B

Labs en C

```
#include <pic.h> // Include header file for MCU
#define _XTAL_FREQ    20000000 // Define Frequency 20.0 MHz for function
__delay_ms
__CONFIG(HS & WDTDIS & LVPDIS);    // Config. High speed clock,Disable
watchdog and Disable LVP
void Delay_ms(unsigned int tick)
{
     while(tick--)        // Loop counter delay time
     {
       __delay_ms(1);  // Delay 1 ms
     }
}
void interrupt INT_SERVICE(void)
{
    if(INTF==1)           // Ensure check INTF flag
    {
        PORTC ^= 0xFF;    // Toggle LED at PORTC
        Delay_ms(10);     // Delay a few time
        INTF = 0;         // Clear INTF
    }
}
```

Labs en C

```
void main()
{
    ANS12 = 0;      // Set RB0 as digital port
    TRISB0 = 1;     // Set RB0 input mode
    TRISC = 0;      // Set PORTC as digital port
    PORTC = 0;      // Ensure clear data of PORTC
    INTF = 0;       // Ensure clear external interrupt flag
    INTE = 1;       // Enable interrupt from INT/RB0
    GIE = 1;           // Enable global interrupt
    while(1);       // Break program

}
```

Ecrire un programme qui permet de faire un décodeur 7 segment entre le port A et le port B