

## كل ما يخص الكاتب

وسام الزعفراني عمري 25 سنة متحصل على الاجازة التطبيقية في علوم الاعلامية ونظم البرمجيات مهتم بعدة مجالات من ابرزها عالم الروبوتكس ملم بجميع أساسيات الكهرباء املك خبرة لا بأس بها اهتم بالتقنية حالياً اشتغل مبرمج ويب وتطبيقات سطح المكتب استطيع إدارة سيرفرات يونكس . مهتم بعلم الفضاء فهدفي يتمثل في بناء روبوت قادر على تصوير تقوس الأرض من الفضاء .



## أهدافي المستقبلية

- ❖ بناء طائرة شراعية ذكية
- ❖ بناء روبوت مائي و المشاركة في المسابقة العالمية التي تقام في النازا
- ❖ النجاح على المستوى العاطفي وتكوين أسرة
- ❖ محاولة السفر إلى البقاع المقدسة إذا سمحت لنا الظروف

## التواصل معي

يمكن التواصل معي على :

البريد الإلكتروني :

zaafraniwissem@gmail.com

الفيسبوك :

<https://www.facebook.com/trez.hochrinada>

تويتر:

<https://twitter.com/ZaafraniWissem>

الهاتف:

+216 26664243



## I LES MICROCONTROLEURS

### 1) Qu'est ce qu'un microcontrôleur :

C'est un ordinateur monté dans un circuit intégré. Les avancées technologiques en matière d'intégration, ont permis d'implanter sur une puce de silicium de quelques millimètres carrés la totalité des composants qui forment la structure de base d'un ordinateur. Leur prix varie de quelques Euros à une dizaine d'Euros pour les plus complexes. Comme tout ordinateur, on peut décomposer la structure interne d'un microprocesseur en trois parties :

- Les mémoires
- Le processeur
- Les périphériques

C'est ce qu'on peut voir sur la figure 1 :

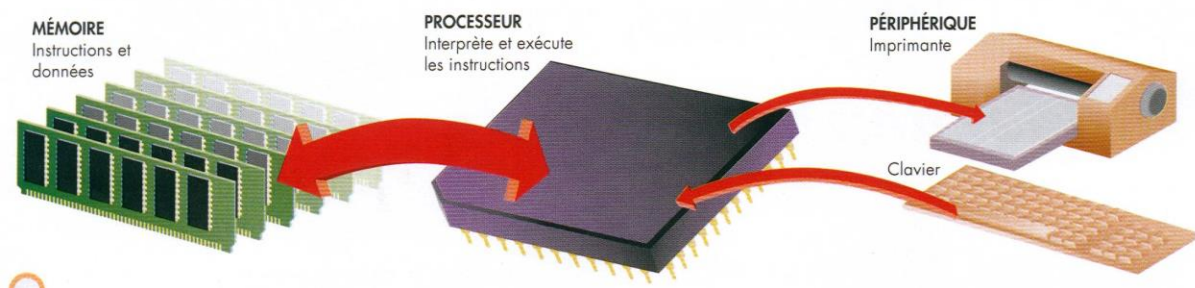


figure 1

- ☛ les mémoires sont chargées de stocker le programme qui sera exécuté ainsi que les données nécessaires et les résultats obtenus
- ☛ le processeur est le cœur du système puisqu'il est chargé d'interpréter les instructions du programme en cours d'exécution et de réaliser les opérations qu'elles contiennent. Au sein du processeur, l'unité arithmétique et logique interprète, traduit et exécute les instructions de calcul.
- ☛ les périphériques ont pour tâche de connecter le processeur avec le monde extérieur dans les deux sens. Soit le processeur fournit des informations vers l'extérieur (périphérique de sortie), soit il en reçoit (périphérique d'entrée).

## 2) Intérêt des microcontrôleurs :

Les microcontrôleurs sont de taille tellement réduite qu'ils peuvent être sans difficulté implantés sur l'application même qu'ils sont censés piloter. Leur prix et leurs performances simplifient énormément la conception de système électronique et informatique. L'utilisation des microcontrôleurs ne connaît de limite que l'ingéniosité des concepteurs, on les trouve dans nos cafetières, les magnétoscopes, les radios ... Une étude menée en l'an 2004 montre qu'en moyenne, un foyer américain héberge environ 240 microcontrôleurs.

## II PRESENTATION GENERALE DU PIC 16F84

### 1) Classification du PIC 16F84

Le PIC 16F84 est un microcontrôleur 8 bits. Il dispose donc d'un bus de données de huit bits. Puisqu'il traite des données de huit bits, il dispose d'une mémoire de donnée dans laquelle chaque emplacement (défini par une adresse) possède huit cases pouvant contenir chacune un bit.

### 2) Architecture interne

La structure générale du PIC 16F84 comporte 4 blocs comme le montre la figure 2 :

- Mémoire de programme
- Mémoire de données
- Processeur
- Ressources auxiliaires ( périphériques )

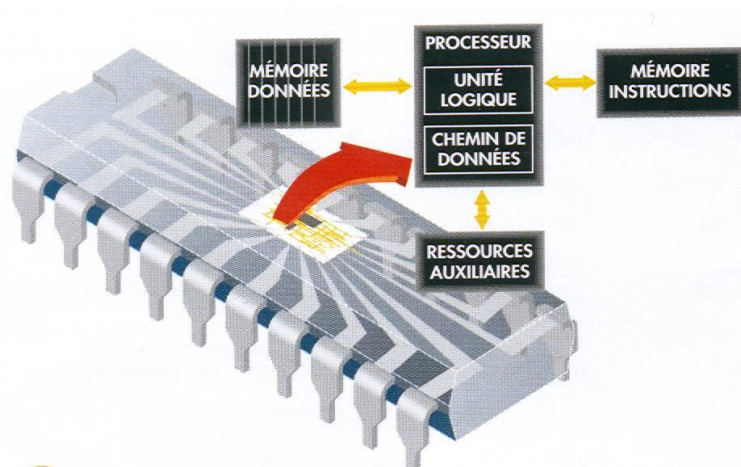


figure 2

☛ La mémoire de programme contient les instructions pilotant l'application à laquelle le microcontrôleur est dédié. Il s'agit d'une mémoire non volatile ( elle garde son contenu, même en l'absence de tension ), elle est de type FLASH c'est à dire qu'elle peut être programmée et effacée par l'utilisateur via un programmeur et un PC. La technologie utilisée permet plus de 1000 cycles d'effacement et de programmation. Pour le PIC 16F84 cette mémoire est d'une taille de 1024\*14 bits, c'est à dire qu'elle dispose de 1024 emplacements ( de

000h à 3FFh) contenant chacun 14 cases car dans le cas du PIC, les instructions sont codées sur 14 bits. On peut donc stocker 1024 instructions.

☛ La mémoire de donnée est séparée en deux parties :

- une mémoire RAM de 68 octets puisque le bus de donnée est de huit bits. Cette RAM est volatile (les données sont perdues à chaque coupure de courant). On peut y lire et écrire des données.

- une mémoire EEPROM de 64 octets dans laquelle on peut lire et écrire des données ( de huit bits soit un octet ) et qui possède l'avantage d'être non volatile ( les données sont conservées même en l'absence de tension ). La lecture et l'écriture dans cette mémoire de données sont beaucoup plus lentes que dans la mémoire de données RAM.

- ☛ Le processeur est formé de deux parties :
  - une unité arithmétique et logique (UAL) chargée de faire des calculs.
  - un registre de travail noté W sur lequel travail l'UAL.
- ☛ les ressources auxiliaires qui sont dans le cas du PIC16F84
  - ports d'entrées et de sorties.
  - temporisateur.
  - interruptions
  - chien de garde
  - mode sommeil

Ces ressources seront analysées dans la suite du cours.

### III STRUCTURE INTERNE DU PIC 16F84

#### 1) Brochage et caractéristiques principales

Le PIC16F84 est un circuit intégré de 18 broches ( figure 3) :

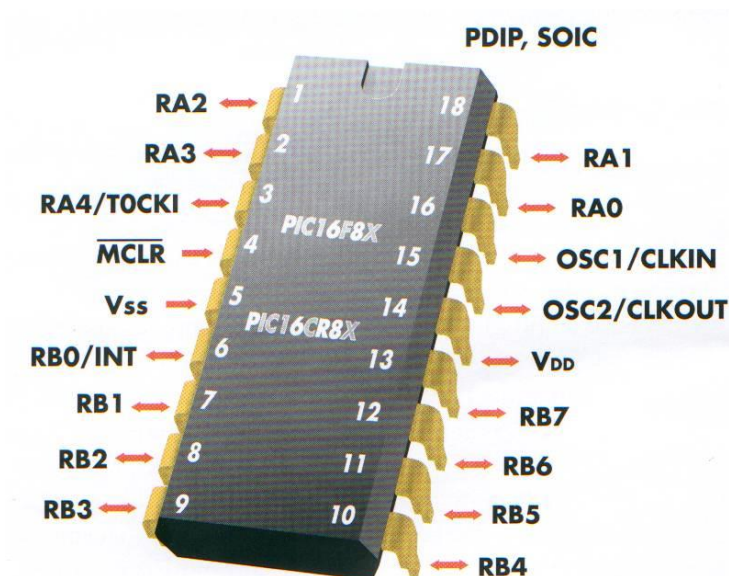


figure 3

- ☛ L'alimentation du circuit est assurée par les pattes VDD et VSS. Elles permettent à l'ensemble des composants électroniques du PIC de fonctionner. Pour cela on relie VSS (patte 5) à la masse ( 0 Volt ) et VDD (patte 14 ) à la

borne positive de l'alimentation qui doit délivrer une tension continue comprise entre 3 et 6 Volts.

☛ Le microcontrôleur est un système qui exécute des instructions les unes après les autres à une vitesse ( fréquence ) qui est fixée par une horloge interne au circuit. Cette horloge doit être stabilisée de manière externe au moyen d'un cristal de quartz connecté aux pattes OSC1/CLKIN (patte 16) et OSC2/CLKOUT ( patte 15 ) . Nous reviendrons en détail sur l'horloge au paragraphe 3.

---

☛ La patte 4 est appelée MCLR. Elle permet lorsque la tension appliquée est égale à 0V de réinitialiser le microcontrôleur. C'est à dire que si un niveau bas ( 0 Volt ) est appliqué sur MCLR le microcontrôleur s'arrête, place tout ses registres dans un état connu et se redirige vers le début de la mémoire de programme pour recommencer le programme au début ( adresse dans la mémoire de programme :0000 ).



---

A la mise sous tension, la patte MCLR étant à zéro, le programme démarre donc à l'adresse 0000, ( MCLR=Master Clear Reset ).

☛ Les broches RB0 à RB7 et RA0 à RA4 sont les lignes d'entrées/sorties numériques. Elles sont au nombre de 13 et peuvent être configurées en entrée ou en sortie. Ce sont elles qui permettent au microcontrôleur de dialoguer avec le monde extérieur (périphériques). L'ensemble des lignes RB0 à RB7 forme le port B et les lignes RA0 à RA4 forment le port A. Certaines de ces broches ont aussi d'autres fonctions (interruption, timer ).

## 2) Structure interne

La structure interne du PIC16F84 est donnée figure 4 : ( structure HARVARD : la mémoire de programme et la mémoire de données sont séparées contrairement à l'architecture Von Neuman qui caractérise d'autres fabricants de microcontrôleurs )

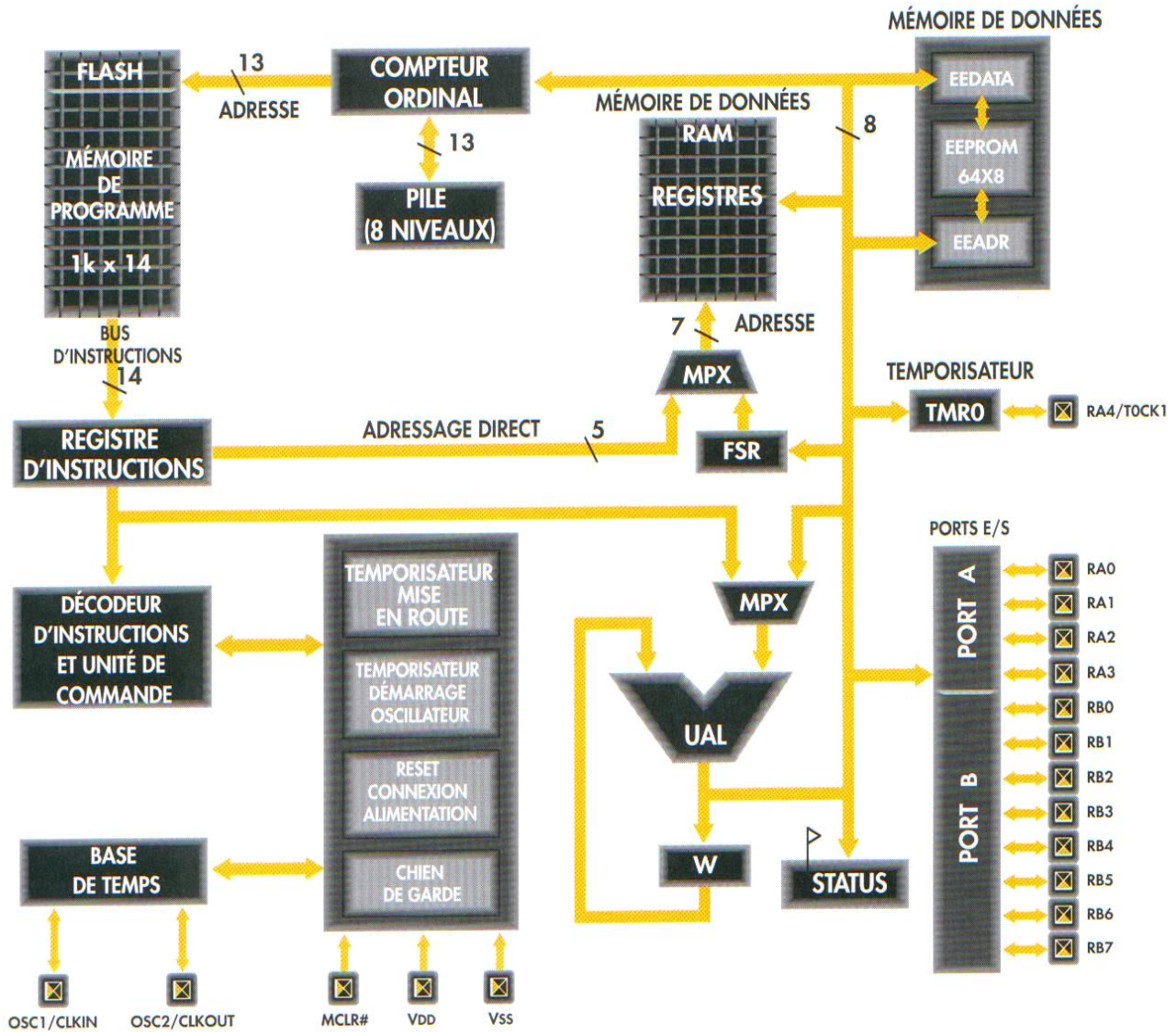


figure 4

On retrouve sur ce schéma la mémoire de programme, la mémoire RAM de données, la mémoire EEPROM, les ports A et B, ainsi que la partie processeur avec l'UAL et le registre de travail W (work). Nous allons étudier à présent plus en détail le fonctionnement du PIC.

### 3) Principe de fonctionnement du PIC

Un microcontrôleur exécute des instructions. On définit « le cycle instruction » comme le temps nécessaire à l'exécution d'une instruction. Attention de ne pas confondre cette notion avec le cycle d'horloge qui correspond au temps nécessaire à l'exécution d'une opération élémentaire ( soit un coup d'horloge ).

Une instruction est exécutée en deux phases :

- la phase de recherche du code binaire de l'instruction stocké dans la mémoire de programme
- la phase d'exécution ou le code de l'instruction est interprété par le processeur et exécuté.

Chaque phase dure 4 cycles d'horloge comme le montre la figure 5 :

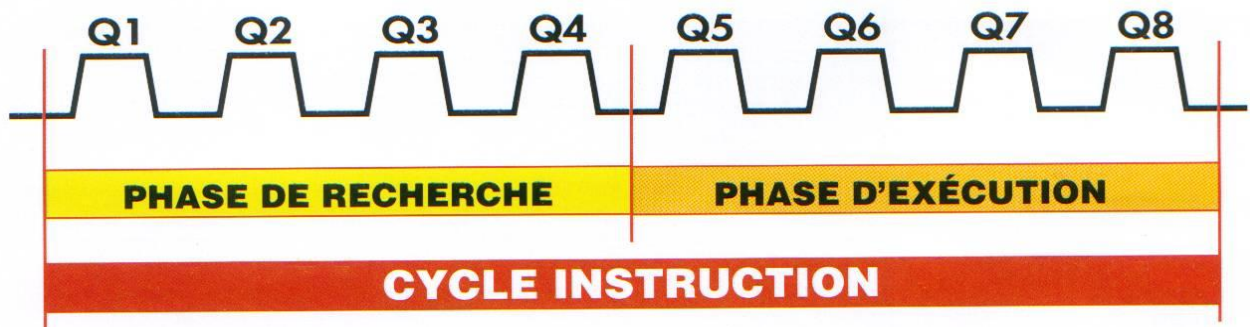


figure 5

On pourrait donc croire qu'un cycle instruction dure 8 cycles d'horloge mais l'architecture particulière du PIC lui permet de réduire ce temps par deux. En effet, comme les instructions issues de la mémoire de programme circulent sur un bus différent de celui sur lequel circulent les données, ainsi le processeur peut effectuer la phase de recherche d'une instruction pendant qu'il exécute l'instruction précédente ( Voir figure 6 et 7 ).

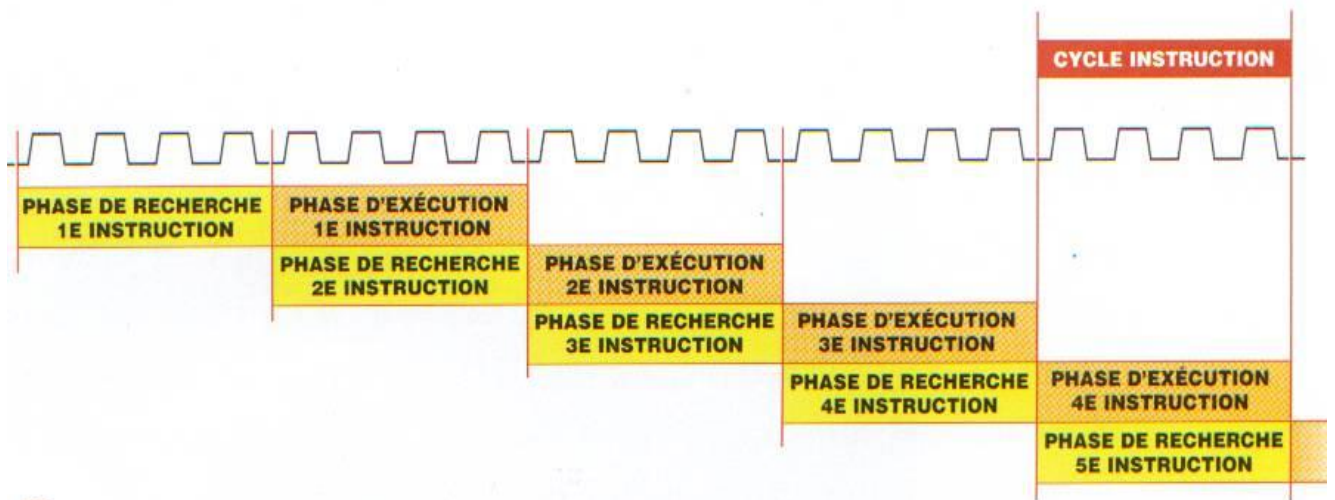


figure 6



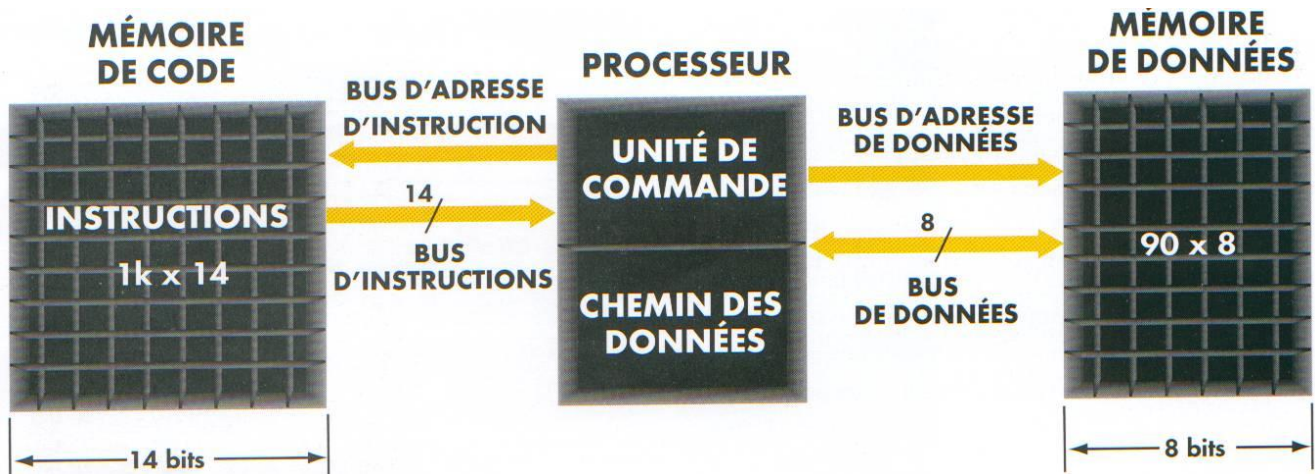


figure 7

#### 4) Déroulement d'un programme

Le déroulement d'un programme s'effectue de façon très simple. A la mise sous tension, le processeur va chercher la première instruction qui se trouve à l'adresse 0000 de la mémoire de programme, l'exécute puis va chercher la deuxième instruction à l'adresse 0001 et ainsi de suite (sauf cas de saut ou d'appel de sous programme que nous allons voir plus loin). On parle de fonctionnement séquentiel. La figure 8 va nous permettre de mieux comprendre le fonctionnement :

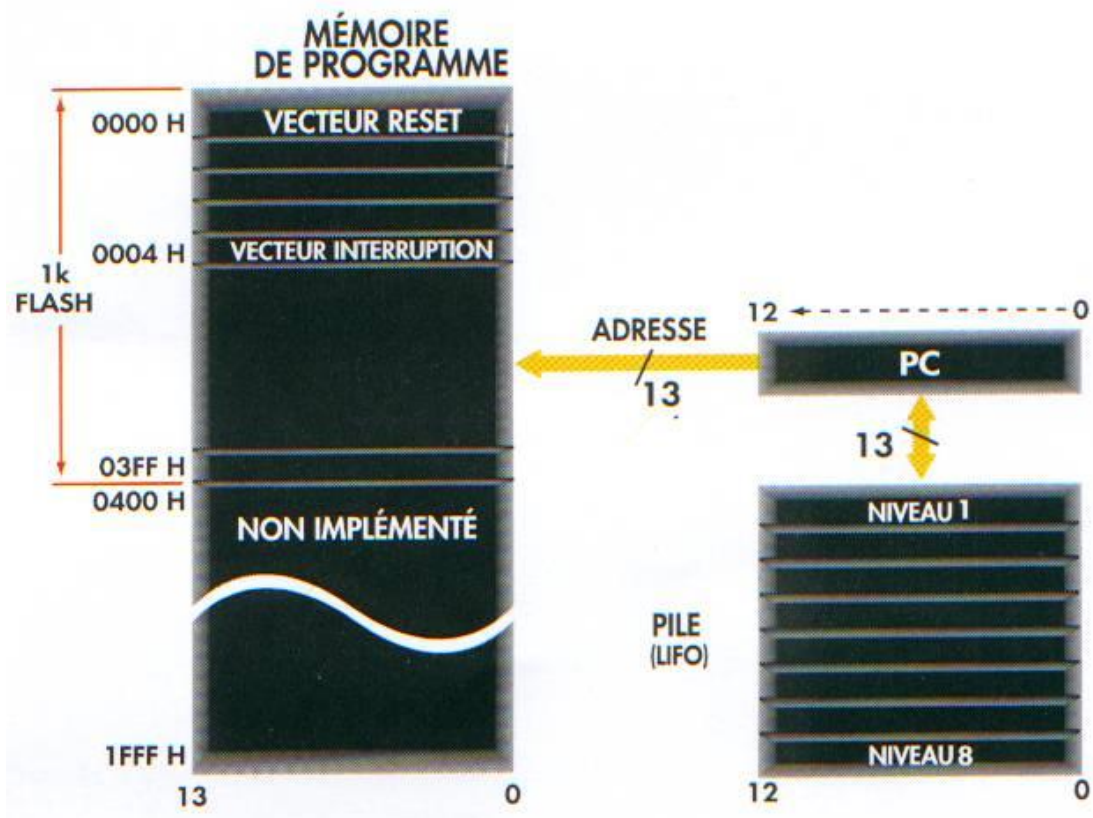


figure 8

- ☛ On constate sur cette figure que la mémoire de programme contient 1024 emplacements ( 3FF en hexadécimale ) contenant 14 bits ( de 0 à 13 ). Une instruction occupe un emplacement qui est défini par une adresse. Le processeur peut alors sélectionner l'emplacement souhaité grâce au bus d'adresse et il peut lire son contenu ( ici l'instruction ) grâce à son bus d'instruction ( voir figure 7 ). Cet adressage s'effectue à l'aide d'un compteur ordinal appelé PC qui lors de la mise sous tension démarre à zéro puis s'incrémente de 1 tous les quatre coups d'horloge, on exécute bien ainsi les instructions les unes à la suite des autres.
- ☛ Mais il arrive que dans un programme on fasse appel à un sous programme dont l'adresse de l'instruction ne se trouve pas juste après celle qui est en train d'être exécutée. C'est le rôle de la pile qui sert à emmagasiner de manière temporaire l'adresse d'une instruction. Elle est automatiquement utilisée chaque fois que l'on appelle un sous programme et elle permet une fois que l'exécution du sous programme est terminée de retourner dans le programme principal juste après l'endroit où l'on a appelé le sous programme. On constate que cette pile possède huit niveaux, cela signifie qu'il n'est pas possible d'imbriquer plus de huit sous programmes, car au-delà de huit, le processeur ne sera plus capable de retourner à l'adresse de base du programme principal.
- ☛ L'adresse 0000 est réservée au vecteur RESET, cela signifie que c'est à cette position que l'on accède chaque fois qu'il se produit une réinitialisation ( 0 volts sur la patte MCLR ). C'est pour cette raison que le programme de fonctionnement du microcontrôleur doit toujours démarrer à cette adresse.
- ☛ L'adresse 0004 est assignée au vecteur d'interruption et fonctionne de manière similaire à celle du vecteur de Reset. Quand une interruption est produite et validée, le compteur ordinal PC se charge avec 0004 et l'instruction stockée à cet emplacement est exécutée.

## 5) La mémoire de données RAM

Si l'on regarde la mémoire de donnée RAM, on s'aperçoit que celle-ci est un peu particulière comme le montre la figure 9 :

On constate en effet que cette mémoire est séparée en deux pages ( page 0 et page 1 ). De plus, on remarque que tant pour la page 0 que pour la page 1, les premiers octets

sont réservés ( SFR pour Special File Register ). Ces emplacements sont en effet utilisés par le microcontrôleur pour configurer l'ensemble de son fonctionnement. On les appelle registres spécifiques et nous verrons au chapitre suivant leurs rôles. Le bus d'adresse qui permet d'adresser la RAM est composé de 7 fils ce qui veut dire qu'il est capable d'adresser 128 emplacements différents. Or, chaque page de la RAM est composée de 128 octets, le bus d'adresse ne peut donc pas accéder aux deux pages, c'est pourquoi on utilise une astuce de programmation qui permet de diriger le bus d'adresse soit sur la page 0, soit sur la page 1. Cela est réalisé grâce à un bit d'un registre spécifique ( le bit RP0 du registre STATUS ) dont nous verrons le fonctionnement plus loin.

La RAM de données proprement dite se réduit donc à la zone notée GPR (Registre à usage générale) qui s'étend de l'adresse 0Ch ( 12 en décimale ) jusqu'à 4Fh ( 79 en décimale ) soit au total 68 registres en page 0 et autant en page 1, mais on constate que les données écrites en page 1 sont redirigées en page 0 cela signifie qu'au final l'utilisateur dispose uniquement de 68 registres ( donc 68 octets de mémoire vive ) dans lesquels il peut écrire et lire à volonté en sachant qu'à la mise hors tension, ces données seront perdues.



# MÉMOIRE DE DONNÉES RAM

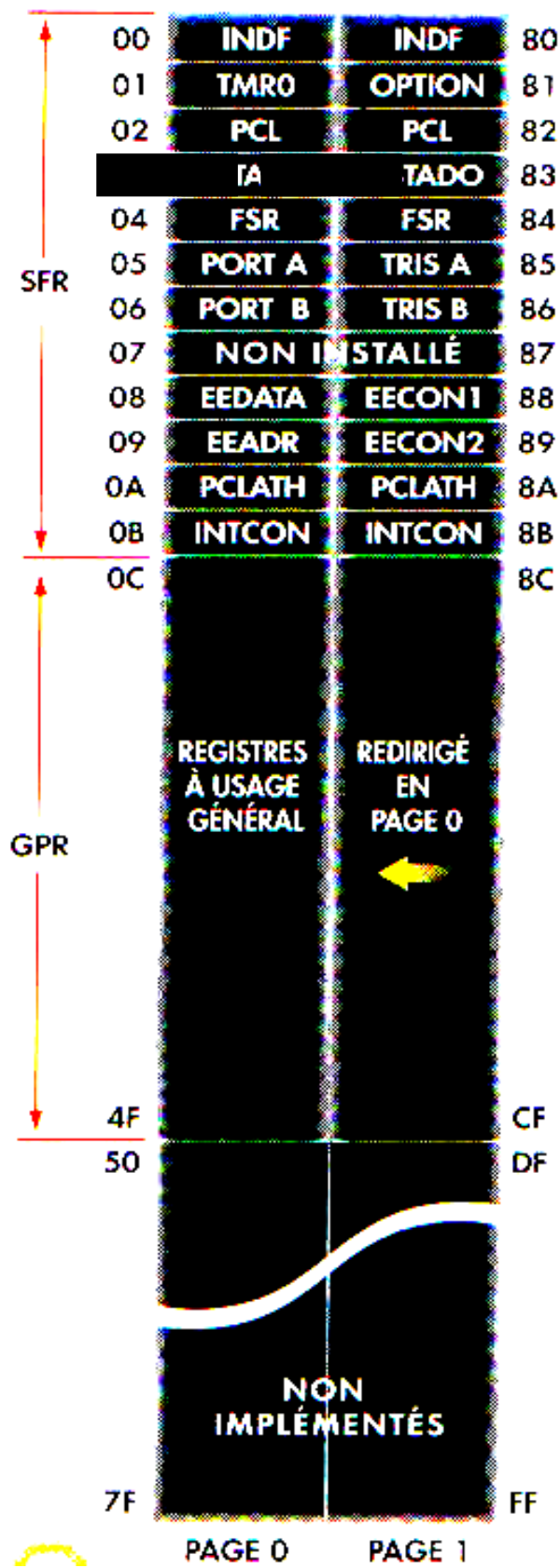


figure 9

## 6) Les registres

Nous avons vu au chapitre précédent que la mémoire de données RAM contenait des registres spécifiques qui permettent de configurer le PIC, nous allons les détailler un à un et voir comment on peut accéder à la page 0 ou la page 1. Afin de faciliter la compréhension, les registres les plus utilisés sont encadrés.

☛ adresse 00 et 80, INDF. Cette adresse ne contient pas de registre physique, elle sert pour l'adressage indirect. (non utilisée dans le projet de cette année)

☛ adresse 01, TMR0. Contenu du Timer (8 bits). Il peut être incrémenté par l'horloge ( $f_{osc}/4$ ) c'est à dire tous les 4 coups d'horloge ou par la broche RA4.

☛ adresse 02 et 82, PCL. 8 bits de poids faibles du compteur ordinal PC. Les 5 (13-8) bits de poids forts sont dans PCLATH.

☛ adresse 03 et 83, STATUS Registre d'état.

les cinq bits de poids faible de ce registre sont en lecture seule, ce sont des témoins (drapeaux ou flag en anglais) caractérisant le résultat de l'opération réalisée par l'UAL. Le bit RP0 est lui en lecture/écriture et c'est lui qui permet de sélectionner la page dans la mémoire RAM.

Si RP0=0 on accède à la page 0 et si RP0=1 on accède à la page 1.

RP0	TO/	PD/	Z	DC	C
-----	-----	-----	---	----	---

Au reset, seul le bit RP0 de sélection de page est fixé (RP0=0 : page 0)

TO/ (Time Out) : débordement du timer WDT

PD/ : (Power Down) caractérise l'activité du chien de garde WDT

Z (zéro) résultat nul pour une opération arithmétique et logique.

DC (digit carry) retenue sur un quartet (4 bits)

C (carry) retenue sur un octet (8 bits).

☛ adresse 04 et 84 , FSR . Registre de sélection de registre : contient l'adresse d'un autre registre (adressage indirect, non utilisé dans le projet )

☛ adresse 05 , PORTA . Ce registre contient l'état des lignes du port A ( voir chapitre sur les ports ).

☛ adresse 06 , PORTB . Ce registre contient l'état des lignes du port B ( voir chapitre sur les ports ).

☛ adresse 08 , EEDATA . Contient un octet lu ou à écrire dans l'EEPROM de données.

☛ adresse 09 , EEADR . Contient l'adresse de la donnée lue ou écrite dans l'EEPROM de données.

☛ adresse 0A et 8A , PCLATH . Voir l'adresse 02 PCL.

☛ adresse 0B et 8B , INTCON .

Contrôle des 4 interrupteurs

GIE	EEI	T0I	INT	RBIE	T0I	INTF	RBI
	E	E	E		F		F

Masques :

**GIE** : (Global Interrupt Enable) : masque global d'inter.

EEIE: (EEPROM Interrupt Enable) autorise l'interruption venant de l'EEPROM.

T0IE: (Timer 0 Interrupt Enable) autorise l'interruption provoquée par le débordement du TIMER0

INTE: ( Interrupt Enable) autorise l'interruption provoquée par un changement d'état sur broche RB0/INT

RBIE: (RB Interrupt Enable) autorise les interruptions provoquées par un changement d'états sur l'une des broches RB4 à RB7.

Si ces bits sont mis à 1 , ils autorisent les interruptions pour lesquels ils sont dédiés .

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.

☛ adresse 81 , OPTION

8 bits (tous à 1 au RESET) affectant le comportement des E/S et des timers.

RBPU/	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0
-------	--------	-----	-----	-----	-----	-----	-----

**RBPU/** (RB Pull Up) Résistances de tirage à Vdd des entrées du port B ( voir le détail du fonctionnement au chapitre port ). Si RBPU/=0 les résistances de pull-up sont connectées en interne sur l'ensemble du port B.



INTEDG (Interrupt Edge) sélection du front actif de l'interruption sur RB0/INT ( 1 pour front montant et 0 pour front descendant ).

RTS (Real Timer Source) sélection du signal alimentant le timer 0 : 0 pour horloge interne, 1 pour RA4/T0CLK

RTE (Real Timer Edge) sélection du 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 ) sélection de la valeur du diviseur de fréquence pour les timers.

➡ adresse 85 , TRISA . Direction des données pour le port A : 0 pour sortir et 1 pour entrer ( voir chapitre sur les ports ).

➡ adresse 86 , TRISB . Direction des données pour le port B : 0 pour sortir et 1 pour entrer ( voir chapitre sur les ports ).

☛ adresse 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

☛ adresse 89 , EECON2 . Register 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.

## 7) Les ports d'entrées/sorties

Le PIC16F84 est équipé de 13 lignes d'entrées/sorties réparties en deux ports :

-le port A : RA0 à RA4

-le port B : RB0 à RB7

Chaque ligne peut être configurée soit en entrée, soit en sortie, et ceci indépendamment l'une de l'autre. Pour cela on utilise les registres TRISA et TRISB . Le bit de poids faible ( b0 ) du registre TRISA correspond à la ligne RA0, le bit b1 de TRISA correspond à RA1 et ainsi de suite. Il en est de même pour le port B et le registre TRISB ( b0 de TRISB correspond à RB0 → b7 correspond à RB7 ). Si l'on veut placer une ligne en sortie il suffit de mettre le bit correspond dans TRISA ou TRISB à 0 (retenez 0 comme Output=sortie) . Si l'on veut placer une ligne en entrée, il suffit de placer le bit correspondant dans TRISA ou TRISB à 1 (retenez 1 comme Input=entrée) .

Les bits des deux registres PORTA et PORTB permettent soit de lire l'état d'une ligne si celle-ci est en entrée, soit de définir le niveau logique d'une ligne si celle-ci est en sortie.

Lors d'un RESET, toutes les lignes sont configurées en entrées.

☛ particularité du portA : les bits b7 à b5 des registres TRISA et PORTA ne correspondent à rien car il n'y a que 5 lignes ( b0 à b4 ) . RA4 est une ligne à collecteur ouvert, cela veut dire que configurée en sortie cette broche assure 0Volt à l'état bas, mais qu'à l'état haut, il est nécessaire de fixer la valeur de la tension grâce à une résistance de tirage (pull up en anglais)

☛ particularité du portB : il est possible de connecter de façon interne sur chaque ligne une résistance de tirage ( pull up ) dont le rôle consiste à fixer la tension de la patte (configuré en entrée) à un niveau haut lorsque qu'aucun signal n'est appliqué sur la patte en question. Pour connecter ces résistances, il suffit de placer le bit RBPU/ du registre OPTION à 0 .

## 8) Le Timer

Dans la majeure partie des applications, il est nécessaire de contrôler le temps; afin de ne pas occuper le microcontrôleur qu'à cette tâche ( boucle de comptage qui monopolise le micro ), on le décharge en utilisant un timer. Le pic 16F84 dispose de deux timers, un à usage général ( le TMR0 ) et un autre utilisé pour le chien de garde ( watch dog WDG ).

Le TMR0 est un compteur ascendant (qui compte) de 8 bits qui peut être chargé avec une valeur initiale quelconque. Il est ensuite incrémenté à chaque coup d'horloge jusqu'à ce que le débordement ait lieu ( passage de FF à 00 ); Le principe est représenté figure 10 :

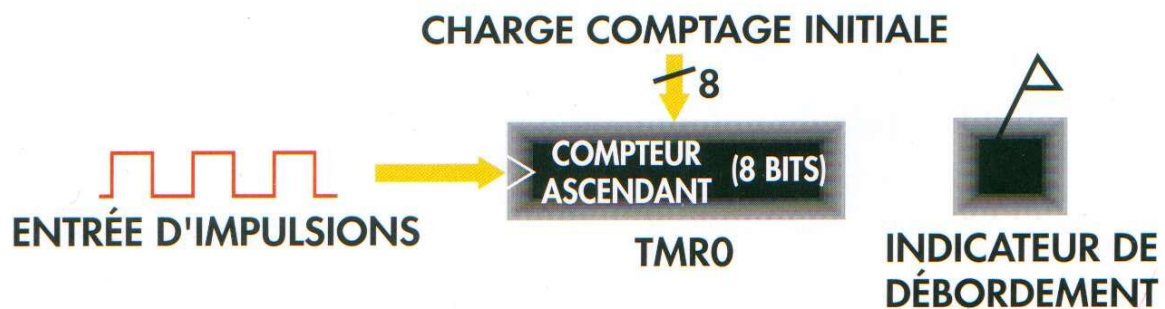


figure 10

Le TMR0 peut remplir deux fonctions:

- Temporisateur ou contrôle du temps. Son entrée d'incrémentation est alors l'horloge qui correspond au cycle instruction (  $F_{osc}/4$  ). Il est possible d'utiliser un pré-diviseur de fréquence que nous verrons plus loin.
  - Compteur d'événements. Dans ce cas les d'impulsions d'entrées du timer sont fournies par la patte RA4/TOCK1
- le choix s'effectue grâce au bit RTS du registre OPTION.

Le pic 16F84 dispose d'un diviseur de fréquence qui peut être assigné soit au chien de garde, soit au TMR0 ( uniquement un à la fois ). L'assignation du pré diviseur se fait



grâce au bit PSA du registre OPTION. La structure interne du TMR0 est donc la suivante ( figure 11 ):

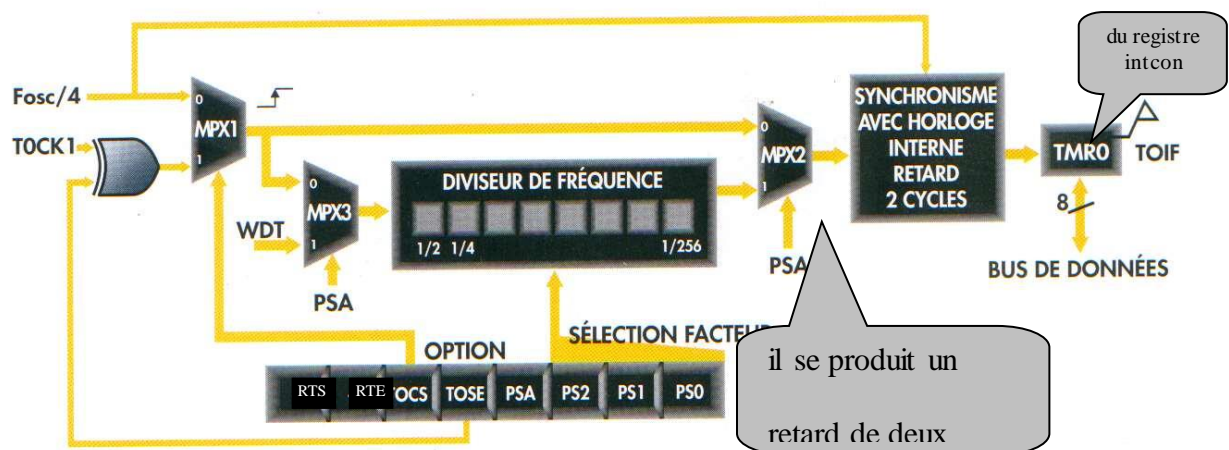


figure 11

Suivant que le pré-diviseur est assigné au chien de garde ou au TMR0, la valeur de la pré-division n'est pas la même, il faut donc être vigilant lors de la programmation comme le montre la figure 12 :

## OPTION

**RBPU#**

**RTS**

**RTE**

**TOSE**

**PSA**

**PS2**

**PS1**

**PS0**

PS2:PS0 Valeur du diviseur de fréquence

PS2	PS1	PS0	Diviseur du TMR0	Diviseur 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

**PSA:** Assignment du diviseur de fréquence

1= Le diviseur de fréquence est assigné au WDT

0= Le diviseur de fréquence est assigné au TMR0

**TOSE:** Type de front sur T0CK1

1= Incrémentation du TMR0 à chaque front descendant

0= Incrémentation du TMR0 à chaque front ascendant

**RTS**

**TOCS:** Type d'horloge pour le TMR0

1= Impulsions introduites via T0CK1 (compteur)

0= Impulsions d'horloge interne Fosc/4 (temporisateur)

**INTEDG**

**INTEDG:** Front actif interruption externe

1= Front ascendant

0= Front descendant

**RPBU/**

**RBPU#:** Résistances de tirage Port B

1= Inhibées

0= Activées

figure 12

Enfin, vu que le timer ne peut que compter, cela oblige à une petite gymnastique lors de l'introduction de la valeur de pré chargement :

exemple :

On veut que le timer nous indique par la mise à un du drapeau T0IF l'écoulement d'une durée de 20ms ( la fréquence d'horloge étant de 4MHz ) d'où  $F_{osc}/4 = 1 \mu s$

si on choisit une pré division de 256 , on aura donc  $20000 \mu s / 256 = 78$

Il ne faut pas charger le TMR0 avec 78 mais avec le complément à deux de cette valeur ( car le timer compte et ne décompte pas ) d'où  $256-78=178$

soit en hexadécimale la valeur B2h à charger dans le registre TMR0.

### 9) Mise en oeuvre

L'utilisation et la mise en oeuvre très simple des PICs les a rendus extrêmement populaire au point que la société qui les fabrique ( MICROCHIP ) est en passe de devenir le leader mondial dans le domaine des microcontrôleurs devant MOTOROLA et INTEL.

Il suffit d'alimenter le circuit par ses deux broches VDD et VSS, de fixer sa vitesse de fonctionnement à l'aide d'un quartz ( figure13) et d'élaborer un petit système pour permettre de réinitialiser le microcontrôleur sans avoir à couper l'alimentation ( figure 14 ).

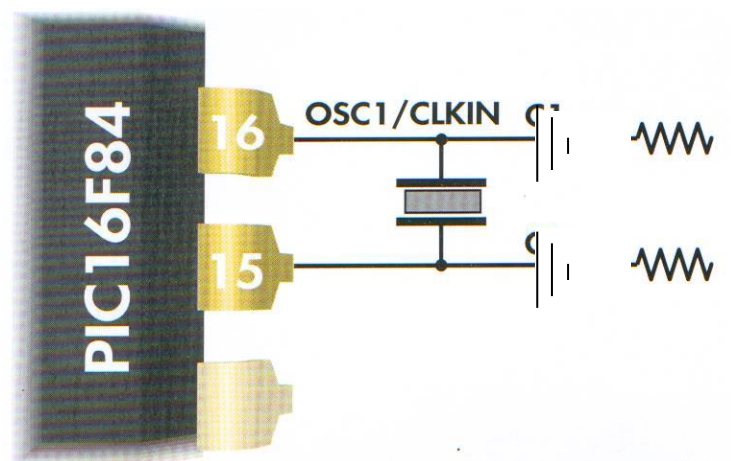


figure 13

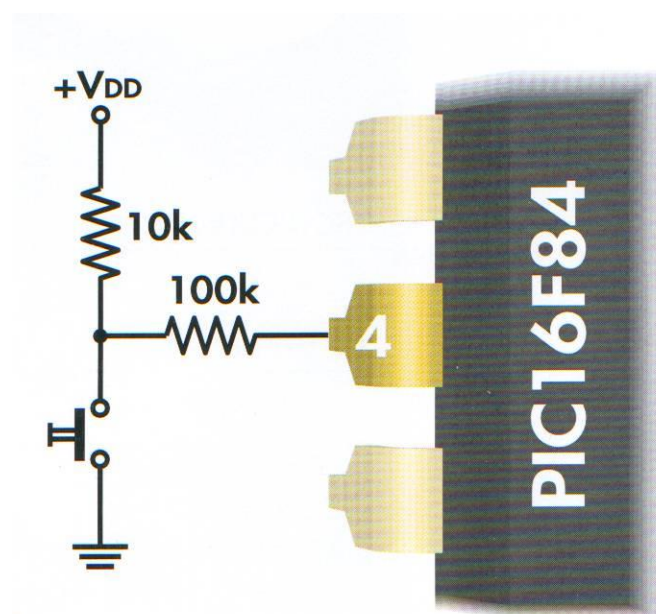


figure 14

Il suffit ensuite d'écrire le programme en langage assembleur ou en C sur un ordinateur grâce au logiciel MPLAB de MICROCHIP ( logiciel gratuit ) puis de le compiler pour le transformer en langage machine et le transférer dans le PIC grâce à un programmeur. Lors de la mise sous tension, tous les registres spécifiques sont placés dans un état déterminé comme le montre la figure 15



PAGE 0

ADR.	NOM	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	VALEUR APRÈS LE RESET	VALEUR AUTRES RESETS
00 h	INDF	Utilise le contenu de FSR pour adresser la mémoire (pas d'existence physique)								----	----
01 h	TMRO	Horloge/compteur en temps réel 8 bits								xxxx xxxx	uuuu uuuu
02 h	PLC	Octet de moindre poids du PC								0000 0000	0000 0000
03 h	STATUS	IRP	IRP1	RPO	TO#	PD#	Z	DC	C	0001 1xxx	000q quuu
04 h	FSR	Adressage indirect avec INDF								xxxx xxxx	uuuu uuuu
05 h	PORT A	-	-	-	RA4/TOCK	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06 h	PORT B	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07 h		Non implémenté, lu comme étant à 0								----	----
08 h	EEDATA	Registre de données EEPROM								xxxx xxxx	uuuu uuuu
09 h	EEADR	Registre d'adresses EEPROM								xxxx xxxx	uuuu uuuu
0A h	PCLATH	-	-	-	S'écrit sur les 5 bits de PCH					---0 0000	---0 0000
0B h	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 0000

PAGE 1

ADR.	NOM	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	VALEUR APRÈS LE RESET	VALEUR AUTRES RESETS
80 h	INDF	Utilise le contenu de FSR pour adresser la mémoire (pas d'existence physique)								----	----
81 h	OPTION	RBPU#	INTDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82 h	PLC	Octet de moindre poids du PC								0000 0000	0000 0000
83 h	STATUS	IRP	IRP1	RPO	TO#	PD#	Z	DC	C	0001 1xxx	000q quuu
84 h	FSR	Adressage indirect avec INDF								xxxx xxxx	uuuu uuuu
85 h	TRISA	-	-	-	Configuration port A					---1 1111	---1 1111
86 h	TRISB	Configuration port B								1111 1111	1111 1111
87 h		Non implémenté, lu comme étant à 0								----	----
88 h	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD	---0 x000	--- q000
89 h	EECON2	Registre de commande EEPROM (pas d'existence physique)								----	----
0A h	PCLATH	-	-	-	Configuration port A					---0 0000	---0 0000
	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u

u=unchanged ; x=unknown ; -=unimplemented ( read as 0 ) ; q=value depends on condition

figure 15

Il ne nous reste plus qu'à voir le jeu d'instruction de programmation en assembleur du PIC et c'est là que réside tout l'intérêt puisqu'il ne dispose que de 35 instructions qui lui permettent de réaliser toutes les tâches.



## IV JEU D'INSTRUCTIONS

Afin de comprendre la fonction de chaque instruction, la notation adoptée pour les données et adresses manipulées par les instructions est fort simple et est la suivante :

- f représente un registre
- b représente un numéro de bit en sachant que 0 correspond toujours au bit de poids faible ( le plus à droite dans le registre )
- k représente une donnée aussi appelé littéral

Un certain nombre d'instructions ( ADDWF, ANDWF , etc.. ) utilise une notation spéciale présentée sous la forme :

ADDWF        f,d      Où f indique le registre et où d peut prendre deux valeurs (0 ou 1), ce qui change le comportement de l'instruction . Si d est à 0, le résultat est placé dans le registre de travail W, la valeur dans le registre f est alors inchangée, alors que si d est à 1, le résultat est placé dans le registre f.

Un autre type d'instruction mérite quelques éclaircissements, ce sont les instructions de branchement conditionnel. Prenons comme exemple :

BTFSC        f,b      Qui va vouloir dire ( Bit Test File Skip if Clear ) qui signifie que l'on va tester le bit b du registre f ( b peut prendre une valeur de 0 à 7 pour un registre 8 bits ) .Il peut alors y avoir deux solutions :

- Soit le bit testé est à 1, donc la condition testée n'est pas réalisée, le programme continue alors son déroulement normalement en séquence avec l'instruction juste en dessous.
- Soit le bit testé vaut 0, donc la condition testée est réalisée et le programme saute l'instruction qui suit le BTFSC dans le programme.

Cette façon de programmer peut paraître étrange, mais avec de l'habitude, elle s'avère très pratique et permet de réaliser des programmes compacts et performants.

Les 35 instructions sont donc les suivantes :

➡ ADDLW ( Add Literal to W )

syntaxe : ADDLW k

Opération :  $W + k \rightarrow W$

Bits d'état du registre STATUS affectés : C, DC, Z

on ajoute au registre de travail la valeur k et on place le résultat dans le registre de travail W

durée : 1 cycle instruction ( 4 cycles d'horloge )

➤ ADDWF (Add W to F )

syntaxe : ADDWF f,d

Opération:  $W+f \rightarrow f$  si  $d=1$  ou  $W+f \rightarrow W$  si  $d=0$

Bits d'état du registre STATUS affectés : C,DC,Z

on ajoute le contenu de W et le contenu de f et on place le résultat dans f si  $d=1$  ou dans W si  $d=0$

durée : 1 cycle instruction ( 4 cycles d'horloge )

➤ ANDLW ( And Literal and W )

syntaxe : ANDLW k

Opération:  $W \text{ ET } k \rightarrow W$

Bit d'état du registre STATUS affecté : Z

on effectue un ET logique entre le contenu de W et le littéral k , on place le résultat dans W

durée : 1 cycle instruction ( 4 cycles d'horloge )

➤ ANDWF ( And W with F )

syntaxe : ANDWF f,d

Opération:  $W \text{ ET } f \rightarrow f$  si  $d=1$  ou  $W \text{ ET } f \rightarrow W$  si  $d=0$

Bit d'état du registre STATUS affecté :Z

on effectue un ET logique entre le contenu de W et le contenu de f , on place le résultat dans W si d=0 ou dans f si d=1

durée : 1 cycle instruction ( 4 cycles d'horloge )

☛ BCF ( Bit Clear F )

syntaxe : BCF f,b

Opération :  $0 \rightarrow b(f)$

Bits d'état du registre STATUS affectés : aucuns

on met à 0 le bit b du registre f

durée : 1 cycle instruction ( 4 cycles d'horloge )

☛BSF ( Bit Set F )

syntaxe : BSF            f,b

Opération:  $1 \rightarrow b(f)$

Bits d'état du registre STATUS affectés :aucuns

on met à 1 le bit b du registre f

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛BTFSC ( Bit Test , Skip if Clear )

syntaxe : BTFSC    f,b

Opération:saut de l'instruction qui suit si  $b(f)=0$

Bits d'état du registre STATUS affectés :aucuns

Si le bit b de f est nul, l'instruction qui suit celle-ci est ignorée et traitée comme un NOP. Dans ce cas et dans ce cas seulement, l'instruction BTFSC demande deux cycles pour s'exécuter.

durée :            1 cycle instruction ( 4 cycles d'horloge ) ou 2 cycles

☛BTFSS ( Bit Test , Skip if Set )

syntaxe : BTFSS            f,b

Opération:saut de l'instruction qui suit si  $b(f)=1$

Bits d'état du registre STATUS affectés :aucuns

Si le bit b de f est à 1, l'instruction qui suit celle-ci est ignorée et traitée comme un NOP. Dans ce cas et dans ce cas seulement, l'instruction BTFSS demande deux cycles pour s'exécuter.

durée : 1 cycle instruction ( 4 cycles d'horloge ) ou 2 cycles

➡ CALL (subroutine Call)

syntaxe : CALL label

Bits d'état du registre STATUS affectés :aucuns

On sauvegarde l'adresse de retour dans la pile puis on appelle le sous programme définit avec l'étiquette label

durée : 2 cycles instruction ( 8 cycles d'horloge )



☛ CLRf ( Clear F )

syntaxe : CLRf      f

Opération : 0 → F

Bit d'état du registre STATUS affecté : Z

On met le contenu du registre f à 0 et on positionne Z

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛ CLRW ( Clear W )

syntaxe : CLRW

Opération : 0 → W

Bit d'état du registre STATUS affecté : Z

On met le contenu du registre W à 0 et on positionne Z

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛ CLRWDt ( Clear WatchDog Timer )

syntaxe : CLRWDt

Opération : 0 → WDT et 0 → pré diviseur du Timer

On met le contenu du registre du timer chien de garde à 0 ainsi que le pré diviseur

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛ COMF ( Complement F )

syntaxe : COMF            f,d

opération :  $/f \rightarrow f$  si  $d=1$  ou  $/f \rightarrow W$  si  $d=0$

Bit d'état du registre STATUS affecté :Z

On complémente le contenu du registre f bit à bit , le résultat est placé dans f si  $d=1$  , dans W si  $d=0$  .

durée :            1 cycle instruction ( 4 cycles d'horloge )

### ☛ DECF ( Decrement F )

syntaxe : DECF                      f,d

opération :  $f-1 \rightarrow f$  si  $d=1$  ou  $f-1 \rightarrow W$  si  $d=0$

Bit d'état du registre STATUS affecté :Z

On diminue le contenu du registre f d'une unité, le résultat est placé dans f si  $d=1$ , dans W si  $d=0$  ( dans ce cas f reste inchangé ).

durée :                      1 cycle instruction ( 4 cycles d'horloge )

### ☛ DECFSZ ( Decrement F ,Skip if Zero )

syntaxe : DECFSZ    f,d

opération :  $f-1 \rightarrow f$  si  $d=1$  ou  $f-1 \rightarrow W$  si  $d=0$  et saut si  $f-1=0$

Bit d'état du registre STATUS affecté :aucun

On diminue le contenu du registre f d'une unité, le résultat est placé dans f si  $d=1$ , dans W si  $d=0$  (dans ce cas f reste inchangé). Si le résultat est nul, l'instruction suivante est ignorée et dans ce cas, cette instruction dure deux cycles.

durée :                      1 cycle instruction ( 4 cycles d'horloge ) ou 2 cycles

### ☛ GOTO ( branchement inconditionnel )

syntaxe : GOTO                      label

Bit d'état du registre STATUS affecté :aucun

On effectue un saut dans le programme pour aller à l'adresse pointé par le label précisé dans GOTO

Durée : 2 cycles ( 8 cycles d'horloge )

➡ INCF (Increment F)

syntaxe : INCF f,d

opération :  $f+1 \rightarrow f$  si  $d=1$  ou  $f+1 \rightarrow W$  si  $d=0$

Bit d'état du registre STATUS affecté : Z

On augment le contenu du registre  $f$  d'une unité, le résultat est placé dans  $f$  si  $d=1$  , dans  $W$  si  $d=0$  ( dans ce cas  $f$  reste inchangé ).

durée : 1 cycle instruction ( 4 cycles d'horloge )

➤ INCFSZ ( Increment F , Skip if Zero )

syntaxe : INCFSZ f,d

opération :  $f+1 \rightarrow f$  si  $d=1$  ou  $f+1 \rightarrow W$  si  $d=0$  et saut si  $f-1=0$

Bit d'état du registre STATUS affecté : aucun

On augmente le contenu du registre f d'une unité, le résultat est placé dans f si  $d=1$ , dans W si  $d=0$  (dans ce cas f reste inchangé). Si le résultat est nul, l'instruction suivante est ignorée et dans ce cas, cette instruction dure deux cycles.

durée : 1 cycle instruction ( 4 cycles d'horloge ) ou 2 cycles

➤ IORLW ( Inclusive Or Literal with W )

syntaxe: IORLW k

opération:  $W \text{ OU } k \rightarrow W$

Bit d'état du registre STATUS affecté : Z

On effectue un OU logique entre le contenu de W et le littéral k , le résultat est placé dans W .

durée : 1 cycle instruction ( 4 cycles d'horloge )

➤ IORWF ( Inclusive Or W with F )

syntaxe : IORWF f,d

opération :  $W \text{ OU } f \rightarrow f$  si  $d=1$  ou  $W \text{ OU } f \rightarrow W$  si  $d=0$

Bit d'état du registre STATUS affecté : Z

On effectue un OU entre le contenu de W et le contenu de f, on place le résultat dans f si d=1, dans W si d=0

durée : 1 cycle instruction ( 4 cycles d'horloge )

➡ MOVF ( Move F )

syntaxe : MOVF            f,d

opération :  $f \rightarrow f$  si d=1 ou  $f \rightarrow W$  si d=0

Bit d'état du registre STATUS affecté : Z

On déplace le contenu de f dans f si d=1 ou de f dans W si d=0. Attention, le déplacement de f dans f semble à priori inutile, mais il permet en fait de tester le contenu de f par rapport à 0 et de positionner le bit Z

durée : 1 cycle instruction ( 4 cycles d'horloge )



☛ MOVLW ( Move Literal to W )

syntaxe : MOVLW k

opération :  $k \rightarrow W$

Bit d'état du registre STATUS affecté : aucun

On charge le contenu de W avec le littéral k

durée : 1 cycle instruction ( 4 cycles d'horloge )

☛ MOVWF ( Move W to F )

syntaxe : MOVWF f

opération :  $W \rightarrow f$

Bit d'état du registre STATUS affecté : aucun

On charge le contenu de f avec le contenu de W

durée : 1 cycle instruction ( 4 cycles d'horloge )

☛ NOP ( No Operation )

syntaxe: NOP

opération: néant

Bit d'état du registre STATUS affecté : aucun

On ne fait que consommer du temps machine ( un cycle dans ce cas )

durée : 1 cycle instruction ( 4 cycles d'horloge )

➡ RETFIE ( Return From Interrupt )

syntaxe : RETFIE

opération : Pile → PC

Bit d'état du registre STATUS affecté : aucun

On charge le compteur ordinal avec la valeur qui se trouve au sommet de la pile pour revenir au programme principal lorsque l'exécution du sous programme est terminée.

durée : 2 cycles instruction ( 8 cycles d'horloge )

➤ RETLW ( Return Literal to W )

syntaxe : RETLW            k

opération : k → W , Pile → PC

Bit d'état du registre STATUS affecté : aucun

On charge le contenu de W avec le littéral k puis on charge le compteur ordinal PC avec la valeur qui se trouve au sommet de la pile effectuant ainsi un retour de sous programme .

durée :            2 cycles instruction ( 8 cycles d'horloge )

➤ RETURN ( Return from subroutine )

syntaxe : RETURN

opération : Pile → PC

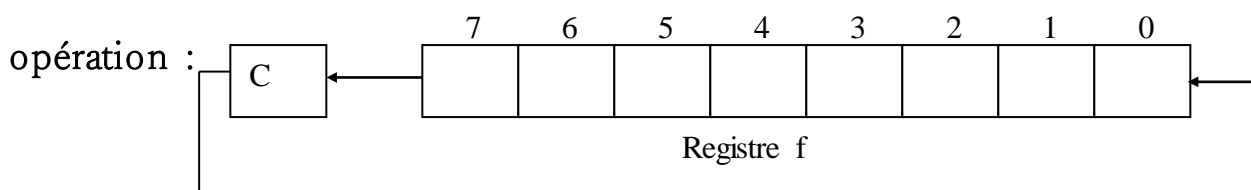
Bit d'état du registre STATUS affecté : aucun

On charge le compteur ordinal PC avec la valeur qui se trouve au sommet de la pile effectuant ainsi un retour de sous programme. C'est un RETLW simplifié .

durée :            2 cycles instruction ( 8 cycles d'horloge )

➤ RLF ( Rotate Left F through carry )

syntaxe : RLF            f,d



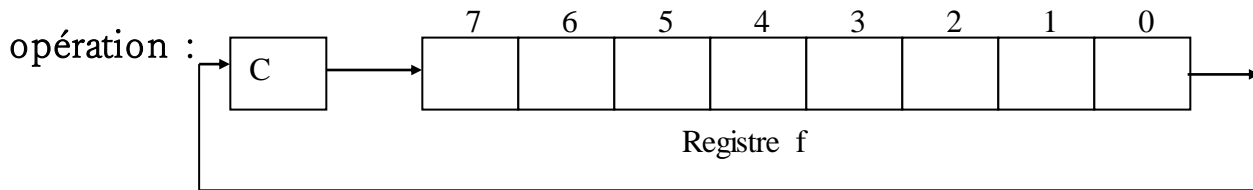
Bit d'état du registre STATUS affecté :C

On effectue une rotation à gauche de un bit du contenu du registre f en passant par le bit de retenu C . Si d=1 le résultat est placé dans f , si d=0 , le résultat est placé dans W

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛ RRF ( Rotate Right F through carry )

syntaxe : RRF            f,d



Bit d'état du registre STATUS affecté :C

On effectue une rotation à droite de un bit du contenu du registre f en passant par le bit de retenu C. Si d=1 le résultat est placé dans f , si d=0, le résultat est placé dans W

durée :            1 cycle instruction ( 4 cycles d'horloge )

☛ SLEEP ( Sleep )

syntaxe : SLEEP

opération : 0 → PD, 1 → T0, 0 → WDT , 0 → pré diviseur

On place le circuit en mode sommeil avec arrêt de l'oscillateur. Cette commande est à utiliser avec précaution, elle nécessite la connaissance du mode sommeil.

☛ SUBLW ( Subtract W from Literal )

syntaxe : SUBLW            k

opération : k-W → W

Bits d'état du registre STATUS affectés :C,DC,Z

On soustrait le contenu du registre W du littéral k et on place le résultat dans W ( soustraction par la méthode du complément à 2 ).

durée : 1 cycle instruction ( 4 cycles d'horloge )

➡ SUBWF ( Subtract W from F )

syntaxe : SUBWF f,d

opération :  $f-W \rightarrow W$  si  $d=0$  ou  $f-W \rightarrow f$  si  $d=1$

Bits d'état du registre STATUS affectés :C,DC,Z

On soustrait le contenu du registre W du contenu du registre f et on place le résultat dans W si  $d=0$ , ou dans f si  $d=1$  ( soustraction par la méthode du complément à 2 ).

durée : 1 cycle instruction ( 4 cycles d'horloge )

## ☛ SWAPF ( Swap F )

syntaxe : SWAPF            f,d

opération :  $f(0-3) \rightarrow f(4-7)$  et  $f(4-7) \rightarrow f(0-3)$  résultat dans W ou f selon d

Bit d'état du registre STATUS affecté : aucun

On échange les quatre bits de poids forts avec les quatre bits de poids faibles et on place le résultat dans W si  $d=0$  , ou dans f si  $d=1$

durée :            1 cycle instruction ( 4 cycles d'horloge )

## ☛ XORLW ( Exclusive Or Literal with W )

syntaxe : XORLW            k

opération :  $W \text{ OU EXCLUSIF } k \rightarrow W$

Bit d'état du registre STATUS affecté : Z

On effectue un OU Exclusif entre W et le littéral k, le résultat est placé dans W

durée :            1 cycle instruction ( 4 cycles d'horloge )

## ☛ XORWF ( Exclusive Or W with F )

syntaxe : XORWF            f,d

opération :  $W \text{ OU EXCLUSIF } f \rightarrow W$  si  $d=0$  ou  $W \text{ OU EXCLUSIF } f \rightarrow f$  si  $d=1$

Bit d'état du registre STATUS affecté : Z

On effectue un OU Exclusif entre W et le contenu de f , le résultat est placé dans W si  $d=0$ , sinon il est placé dans f .



durée : 1 cycle instruction ( 4 cycles d'horloge )

# JEU D'INSTRUCTION

MNÉMONIQUES	PARAMÈTRES	EXPLICATION	ACTIVATION DRAPEAUX
Instructions qui manipulent des registres			
addwf	f, d	ADDITIONNE W et f	C, DC, Z
andwf	f, d	AND de W et f	Z
clrf	f	MISE à 0 de f	Z
clrw		MISE à 0 de W	Z
comf	f, d	COMPLÈMENT de f	Z
decf	f, d	DÉCRÈMENT de f	Z
incf	f, d	INCRÈMENT de f	Z
iorwf	f, d	OU entre W et f	Z
movf	f, d	DÉPLACEMENT de f	Z
movwf	f	DÉPLACEMENT de W dans f	
rlf	f, d	ROTATION à gauche avec retenue	C
rrf	f, d	ROTATION à droite avec retenue	C
subwf	f, d	SOUSTRAIT W de f	C, DC, Z
swapf	f, d	ÉCHANGE les quatre bits de poids fort et les 4 bits de poids faible	
xorwf	f, d	OU exclusif entre W et f	Z
Instructions qui manipulent des bits			
bcf	f, b	MISE à 0 bit b de f	
bsf	f, b	MISE à 1 bit b de f	
Instructions qui manipulent des opérandes immédiats			
addlw	k	ADDITIONNE literal à W	C, DC, Z
andlw	k	AND entre literal et W	Z
iorlw	k	OR entre literal et W	Z
movlw	k	DÉPLACEMENT de literal dans W	
sublw	k	SOUSTRAIT W de literal (K-W))	C, DC, Z
xorlw	k	OU exclusif entre literal et W	Z
Instructions de saut			
btfsc	f, b	TEST du bit b de f. SAUT si 0	
btfss	f, b	TEST du bit b de f. SAUT si 1	
decfsz	f, d	DÉCRÈMENT de f. ; SAUT si 0	
incfsz	f, d	INCRÈMENT de f. ; SAUT si 0	
Instructions de commande et spéciales			
call	k	APPEL à sous-programme	
clrwdt		MISE à 0 du WDT	TO#, PD#
goto	k	BRANCHEMENT sur une adresse	
nop		AUCUNE OPÉRATION	
retfie		RETOUR d'interruption	
retlw	k	RETOUR et charge de W avec literal	
return		RETOUR de sous-programme	
sleep		MISE du microprocesseur en stand-by ou sommeil	TO#, PD#

## V Petite application

Afin de mettre en pratique des notions encore toutes fraîches, il peut être intéressant de clôturer ce cours par une séance de travaux dirigés. Commençons par le commencement avec un petit exemple qui n'a d'autre intérêt que d'appréhender de façon simple un premier programme écrit en assembleur.

- Voici donc un exemple de programme dont la seule fonction est de recopier sur la patte RA0 (donc programmée en sortie) l'état de la patte RB0 (donc programmée en entrée). Sur ce petit exemple, on peut alors aborder les notions d'algorithme et de structure de programmation avec les instructions spécifiques aux logiciels de compilation (org, equ, end, les étiquettes, etc...).

;définition des différentes étiquettes

;pour une structure de programme plus claire

;equ n'est pas une instruction PIC mais sert

;au compilateur

statusequ h'0003';adresse du registre (page 0)

portaequ h'0005';adresse porta (page0)

portbequ h'0006';adresse port b (page 0)

trisa equ h'0085';adresse trisa (page 1)

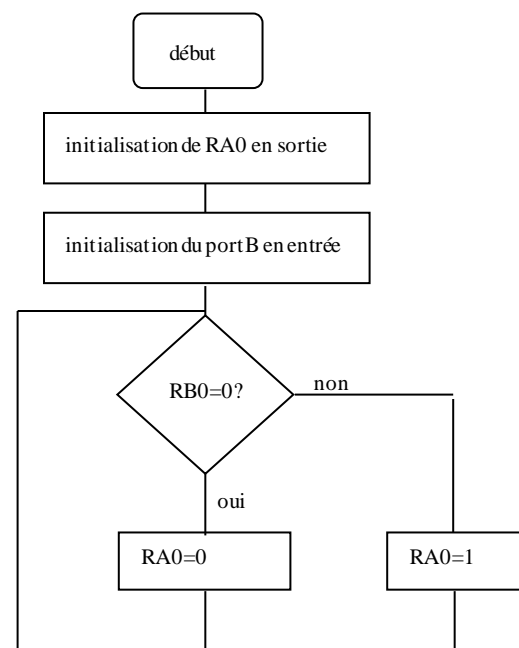
trisb equ h'0086';adresse trisb (page 1)

RA0 equ h'0000';rang du bit RA0 dans porta

RB0 equ h'0000';rang du bit RB0 dans portb

RP0 equ h'0005';rang du bit RP0 dans status

Algorithme du programme



;début du programme indiqué par l'instruction

;org qui est nécessaire au compilateur

org 0000 ;début du programme

;nécessairement à l'adresse

;0000 pour tous les PICs

;initialisation de RA0 en sortie

bsf status,RP0;passage en page 1 de

```

                                ;la mémoire de données
    bcf  trisa,RA0 ;mise à 0 de RA0 (sortie)
;initialisation de RB0 en entrée
    bsf  trisb,RB0 ;mise à 1 de RB0 (entrée)
                                ,mais inutile en pratique
                                ;car réalisé après un RESET
;programme principal
;test de RB0 et recopie en RA0
bouc bcf  status,RP0;retour en page 0
    btfss portb,RB0 ;test RB0 saute l'instruction si RB0=1
    goto eteind      ;va à l'étiquette eteint
    goto allum       ;va à l'étiquette allum

;mise à 0 de RA0
eteintbcf  porta,RA0 ;mise à 0 de RA0
    goto bouc        ;retour à la boucle de test

;mise à 1 de RA0
allumbcf  porta,RA0 ;mise à 1 de RA0
    goto bouc        ;retour à la boucle de test

;fin du programme
;utilisé par le compilateur

    End

```

# LES MICROCONTRÔLEURS

☺ **Activité de découverte:** La signalisation lumineuse d'un robot est assurée par une série de 8 diodes LED de couleur rouge, ces diodes permettent à l'utilisateur de connaître l'état de robot :

Le fonctionnement des diodes est le suivant :

- À l'arrêt les diodes sont toutes éteintes.
- En fonctionnement le robot allume les diodes l'une après l'autre (chenillard).
- Lorsque sa batterie est à la limite de la décharge, le robot effectue un clignotement des diodes.
- Durant la charge de la batterie toutes les diodes sont constamment allumées.

On peut mettre en œuvre la fonction signalisation lumineuse du robot en utilisant un microcontrôleur comme suit :

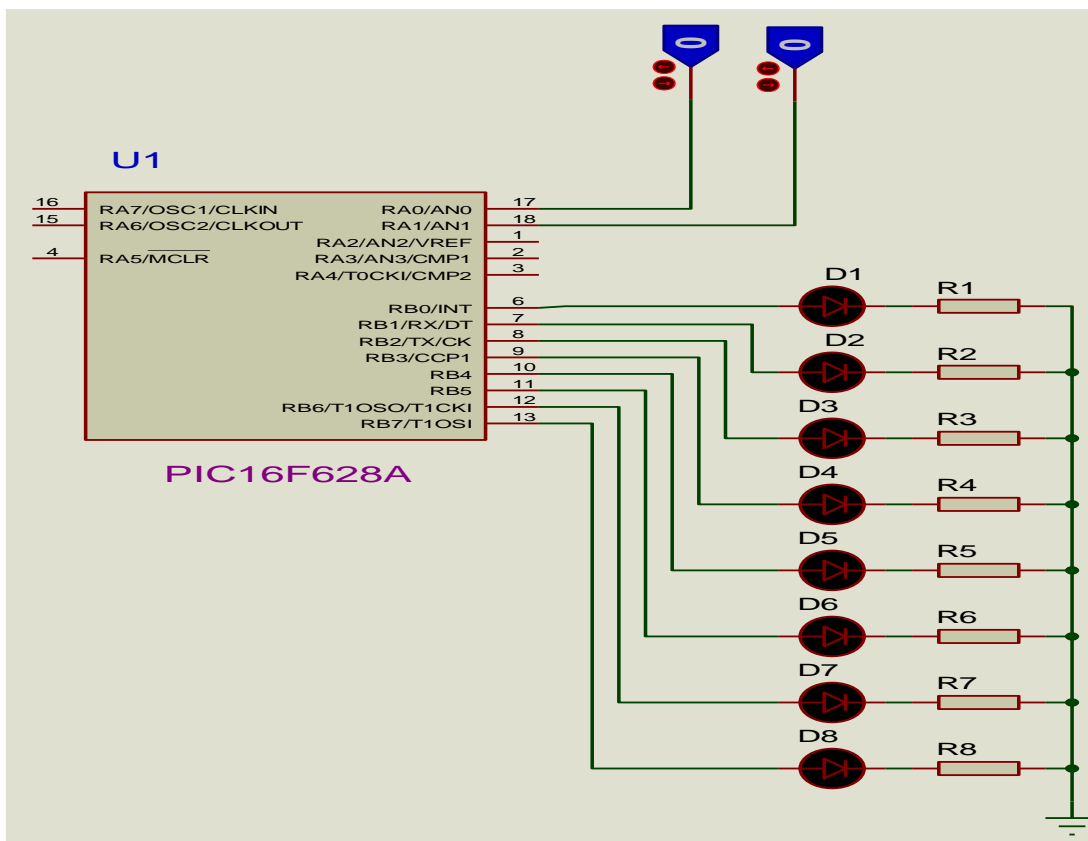




Tableau résumant le fonctionnement des diodes

PORTA	RA1	RA0	Etat du robot	Etat des diodes
0	0	0	Arrêt	Eteintes
1	0	1	Batterie à la limite de décharge	Clignotent
2	1	0	Fonctionne	chenillard
3	1	1	Charge de la batterie	Allumées

✓ Qu'appelle t-on le circuit U1 ? : **Microcontrôleur PIC 16F628A**

✓ Le nombre de broches d'entrée/sortie est : **16**

- ✓ Quelles sont les broches qui sont utilisées comme entrée ? :

RA0 et RA1

- ✓ Quelles sont les broches qui sont utilisées comme sorties ? : RB0

,RB1,RB2,RB3,RB4,RB5,RB6,RB7

- ✓ Les registres TRIS ont pour fonction de configurer les broches d'entrées/sorties des ports.

Chaque broche de chaque port peut être utilisée en entrée ou en sortie :

Un bit à 0 programme la broche correspondante en sortie.

Un bit à 1 programme la broche correspondante en entrée.

Exemple : Remplir le tableau suivant :

TRISB	1	0	1	0	0	1	1	0	= (C6) <sub>HEX</sub>	= (198) <sub>10</sub>
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0		
	entrée	sortie	entrée	sortie	sortie	entrée	entrée	sortie		

- ✓ Déduire alors les valeurs des TRISA et TRISB du microcontrôleur pour l'exemple précédent :

TRISA	1	1	1	1	1	1	1	1	= (11111111) <sub>Bin</sub>	= (FF) <sub>HEX</sub>	= ( 255 ) <sub>10</sub>
-------	---	---	---	---	---	---	---	---	-----------------------------	-----------------------	-------------------------

TRISB	0	0	0	0	0	0	0	0	$= (00000000)_{\text{Bin}}$	$= (00)_{\text{HEX}}$	$= (0)_{10}$
-------	---	---	---	---	---	---	---	---	-----------------------------	-----------------------	--------------

Mise en œuvre d'une application à base de microcontrôleur :

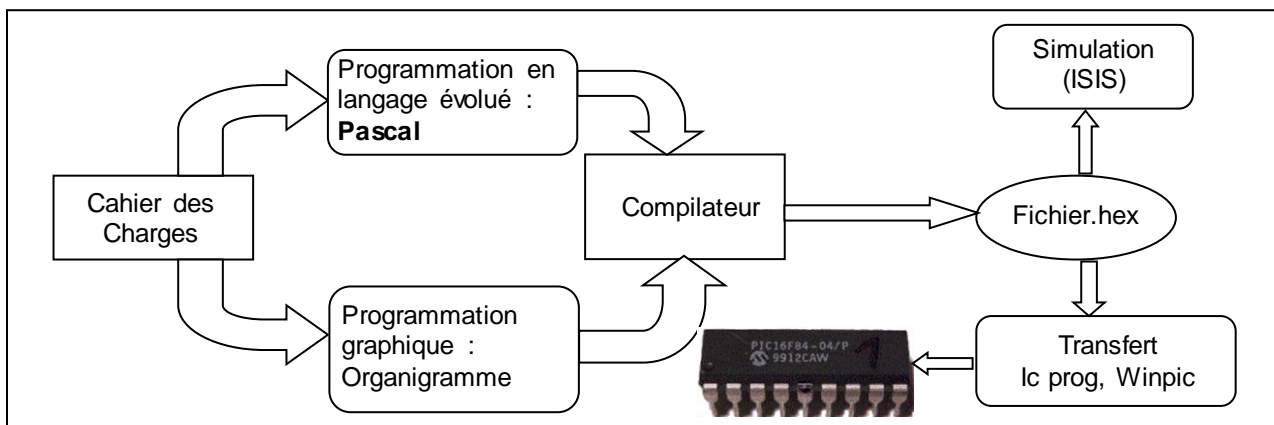
Cette opération consiste à traduire un cahier des charges en un programme codé, puis à le transférer vers la mémoire programme du microcontrôleur.

Divers outils de développement sont mis à la disposition du concepteur. Parmi ces outils, on cite :

- La programmation graphique, elle est basée sur l'interconnexion graphique de symbole ou modules « algorigrammes, grafcet et autres... ».
- La programmation mettant en œuvre un langage évolué tels que langage C, Basic, Pascal,...

Que ce soit par la méthode graphique ou en langage évolué, l'écriture du programme ainsi que

sa mise au point doivent suivre le diagramme suivant :



## I - PROGRAMMATION EN MIKROPASCAL :

### 1 - Structure d'un programme :



Un programme est un texte que le compilateur va traduire en fichier hexadécimal. Le texte d'un programme contient au moins trois parties.

a- L'entête : commence par le mot réservé " *Program* " suivi par le nom du projet.

b- Les déclarations : On déclare les variables, les Procédures et les fonctions utilisées dans le programme.

c- Le corps du programme : Commence par " *Begin* " et se termine par " *End* " suivi d'un point final. Entre " *Begin* " et " *End* " se trouvent les instructions à effectuer par le programme.

## 2 - Les règles de bases :

Algorithmique	Langage PASCAL	Commentaires
<b>Algorithme</b> Nom Algorithme ;	<b>Program</b> Nom programme ;	// Entête
<b>Variables</b>	<b>Var</b>	} // Déclaration
Nom variable : Type ;	Nom variable : Type ;	
<b>Constantes</b>	<b>Const</b>	
Nom constante : Type =valeur ;	Nom constante : Type =valeur ;	} // Programme principal
<b>Début</b>	<b>Begin</b>	
.....	.....	
.....	.....	
<b>Fin.</b>	<b>End.</b>	

a- Toutes instructions ou actions se terminent par un point virgule ;

b- Une ligne de commentaires doit commencer par "{" et se terminer par "}" ou commence par "//".

c- Un bloc d'instructions commence par "Begin" et se termine par "End".

## 3 - Les types de variables utilisées en Mikropascal :

Type	Désignation	Taille	Rang
Bit	bit	1 bit	0 ou 1
Bit registre	sbit	1 bit	0 ou 1
octet	byte	8 bits	0 - 255
Caractère ASCII	char	8 bits	0 - 255
mot	word	16 bits	0 - 65535
Octet signé	short	16 bits	-128 à +127
Entier	integer	16 bits	-32768 à +32767
Entier long	longint	32 bits	-2147483648 à +214783647
Réel	real	32 bits	$\pm 1.17549435082 \times 10^{-38}$ à $\pm 6.80564774407 \times 10^{38}$
Tableau	Array[n]of type	n éléments	Rang du type
Chaîne de caractères	String[N]	N caractères	0 - 255

#### 4 - Les bases du compilateur Mikropascal :

Le décimal

A = 12

L'hexadécimal

A = \$0C ou A = 0x0C

Le binaire

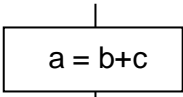
A = %0001100

#### 5 - les opérateurs arithmétiques et logiques :

Opérateurs arithmétiques		Opérateurs de comparaison		Opérateurs logiques	
Opérateur	opération	Opérateur	Opération	Opérateur	Opération
+	Addition	=	Egalité	AND	ET
-	Soustraction	< >	Différent	OR	OU
*	Multiplication	>	Supérieur	XOR	OU exclusif
/	Division	<	inférieur	NOT	NON
div	Division entière	<=	Inférieur ou égale	SHL	Décalage à gauche
mod	Reste de la division	>=	Supérieur ou égale	SHR	Décalage à droite

#### 6 - Les structures usuelles :

a- L'affectation : C'est l'action d'attribuer une valeur à une variable.

Langage graphique	Langage algorithmique	Langage PASCAL
	$a \leftarrow b+c$	$a := b+c$

## b- Les structures alternatives :

Langage graphique	Langage algorithmique	Langage PASCAL
<pre> graph TD     Start(( )) --&gt; Condition{Condition}     Condition -- oui --&gt; Traitement[Traitement]     Condition -- non --&gt; Join(( ))     Traitement --&gt; Join     Join --&gt; Exit(( ))         </pre>	<p>SI condition Alors</p> <p>DEBUT</p> <p>traitement ;</p> <p>FINSI ;</p>	<p>IF condition THEN</p> <p>BEGIN</p> <p>traitement ;</p> <p>END ;</p>
<pre> graph TD     Start(( )) --&gt; Condition{Condition}     Condition -- oui --&gt; T1[Traitement 1]     Condition -- non --&gt; T2[Traitement 2]     T1 --&gt; Join(( ))     T2 --&gt; Join     Join --&gt; Exit(( ))         </pre>	<p>SI condition Alors</p> <p>DEBUT</p> <p>traitement 1 ;</p> <p>FIN</p> <p>SINON</p> <p>DEBUT</p> <p>traitement 2 ;</p> <p>FINSI ;</p>	<p>IF condition THEN</p> <p>BEGIN</p> <p>Traitement 1 ;</p> <p>END</p> <p>ELSE</p> <p>BEGIN</p> <p>traitement 2 ;</p> <p>END;</p>
<pre> graph TD     Start(( )) --&gt; V1{Valeur 1}     V1 -- oui --&gt; A1[Action 1]     V1 -- non --&gt; V2{Valeur 2}     V2 -- oui --&gt; A2[Action 2]     V2 -- non --&gt; V3{Valeur 3}     V3 -- oui --&gt; A3[Action 3]     V3 -- non --&gt; Exit(( ))     A1 --&gt; Exit     A2 --&gt; Exit     A3 --&gt; Exit         </pre>	<p>SELON expression</p> <p>Valeur 1 : action1 ;</p> <p>Valeur 2 : action2 ;</p> <p>..... ;</p> <p>Valeur n : action n ;</p> <p>Autrement : action 0 ;</p> <p>FINSELON ;</p>	<p>CASE expression OF</p> <p>Valeur 1 : action1 ;</p> <p>Valeur 2 : action2 ;</p> <p>..... ;</p> <p>Valeur n : action n ;</p> <p>ELSE : action 0 ;</p> <p>END ;</p>

## c- Les structures itératives ou répétitives :

Langage graphique	Langage algorithmique	Langage PASCAL
<pre> graph TD     Init[i=valeur initiale] --&gt; Body[ ]     Body --&gt; LoopBack(( ))     LoopBack --&gt; Body         </pre>	<p>i : entier ;</p> <p>POUR i variant de « valeur initiale »</p>	<p>i : integer ;</p> <p>FOR i :- « valeur initiale »</p>

## II – APPLICATIONS À BASE DE PIC :

### Activité N° 1: Equations

Objectif : L'objectif de cette activité est de traduire des équations logiques d'un système combinatoire en  
un programme en Mikropascal.

$$S1 = A + B$$

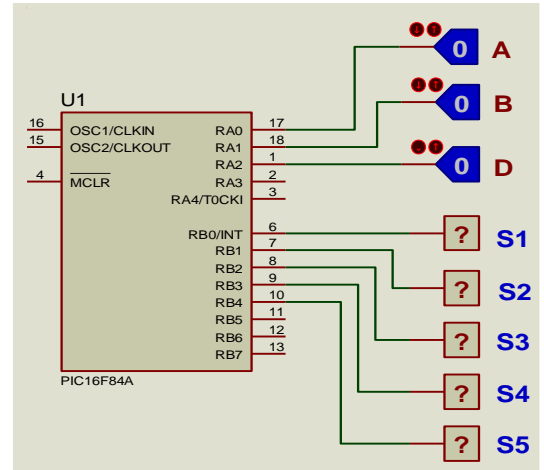
$$S2 = B . D$$

$$S3 = A \oplus B$$

$$S4 = \bar{A} . B + D$$

$$S5 = A \odot B$$

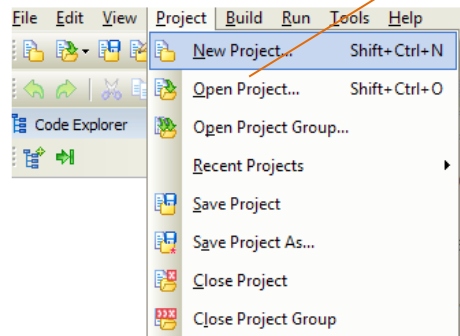
Affectation des entrées/sorties			
Entrées		Sorties	
A	RA0	S1	RB0
B	RA1	S2	RB1
D	RA2	S3	RB2
		S4	RB3
		S5	RB4



Lancer le logiciel *mikropascal*  *pro* et créez un nouveau projet :

1- Dans le menu principal cliquer sur «Project» par suite sur «New project»

2- Une fenêtre «New Project Wizard» apparait, cliquer sur «Next».



la



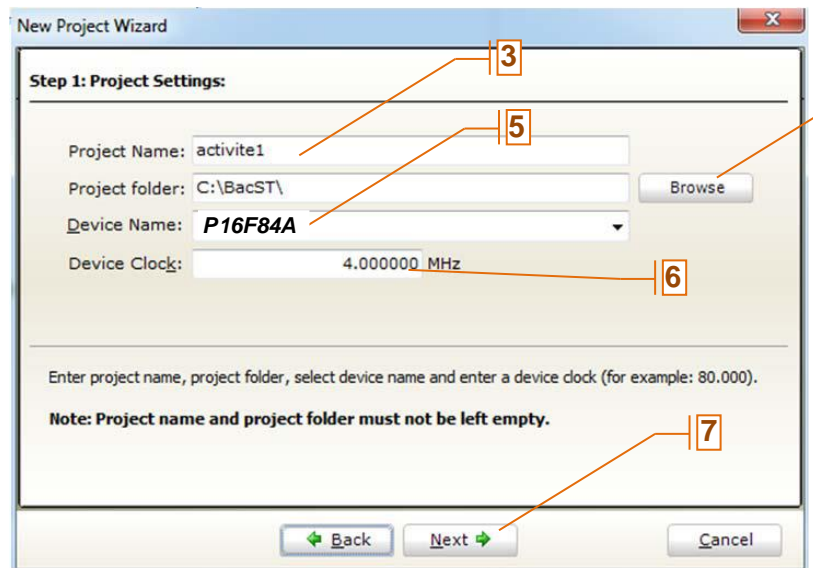
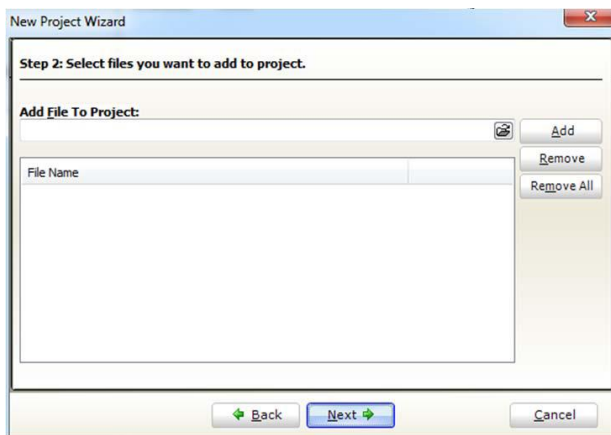
3- Dans le champ «Project Name» Saisir le nom du projet Par exemple : *activitel*, ce nom ne doit pas contenir des caractères accentués ou des espaces.

4- Cliquer sur le bouton « Browse » et sélectionner un répertoire sur le disque de votre PC, ce répertoire contiendra tous les fichiers de votre projet.

5- Sélectionner le type du microcontrôleur : *P16F84A*

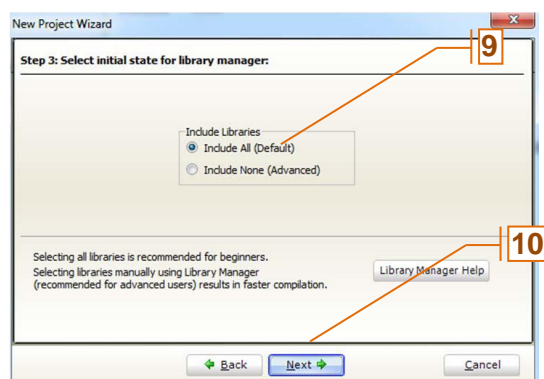
6- Fixer la valeur de l'horloge du microcontrôleur : 4MHz.

7- Cliquer sur le bouton « Next »



8- Cliquer sur le bouton « Next »

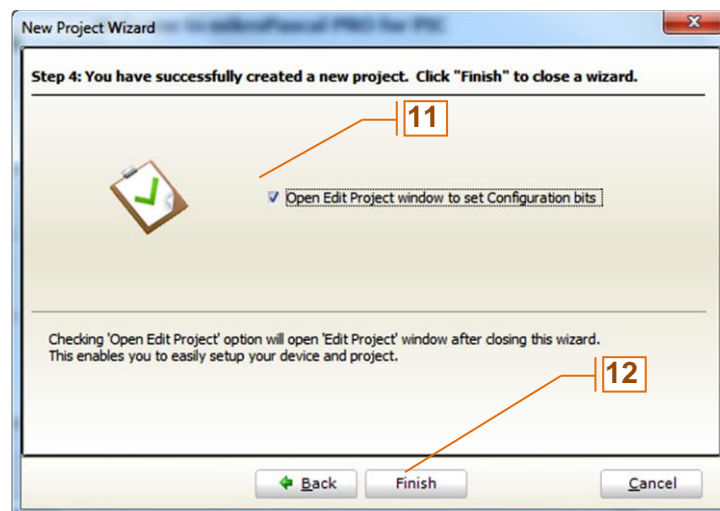
9- Cocher la case « IncludeAll » cette option permet d'utiliser les fonctions prédéfinies de Mikropascal



10- Cliquer sur le bouton

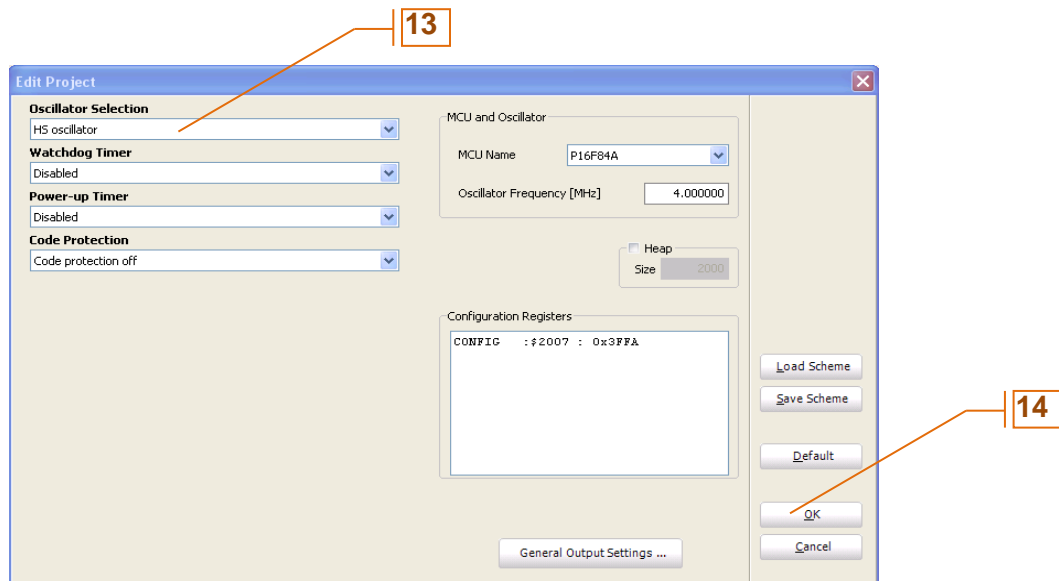
« Next

11- Cocher la case « Open Edit Project window to set Configuration bits ».



12- Cliquer sur le bouton « Finish ».

13- Assurer les réglages des bits de configurations comme indiqué dans la figure suivante.



14- Cliquer sur le bouton « OK ».

15- Compléter puis saisir le programme ci-contre

```
program equations;
```

```
// Nom du programme
```

```
var
```

```
// déclaration des variables
```



```
A: sbit at RA0_bit;    // La variable A est un bit affecté à la broche RA0
B: sbit at RA1_bit;    // La variable B est un bit affecté à la broche RA1
D: sbit at RA2_bit;    // La variable D est un bit affecté à la broche RA2
S1: sbit at RB0_bit;    // La variable S1 est un bit affecté à la broche RB0
S2: sbit at RB1_bit;
S3: sbit at RB2_bit;
S4: sbit at RB3_bit;
S5: sbit at RB4_bit;
```

```
begin
```

```
    TRISA:= $FF;        // Tout le portA est configuré en entrée
```

```
    TRISB:= 0;           // Tout le portB est configuré en sortie
```

```
    PORTB:= 0;          // Initialisation des sorties
```

```
        while true do    // Boucle infinie
```

```
            begin
```

```
                S1:= A Or B;        // equation de S1
```

```
                S2:= B And D;
```

```
                S3:= A xor B ;
```

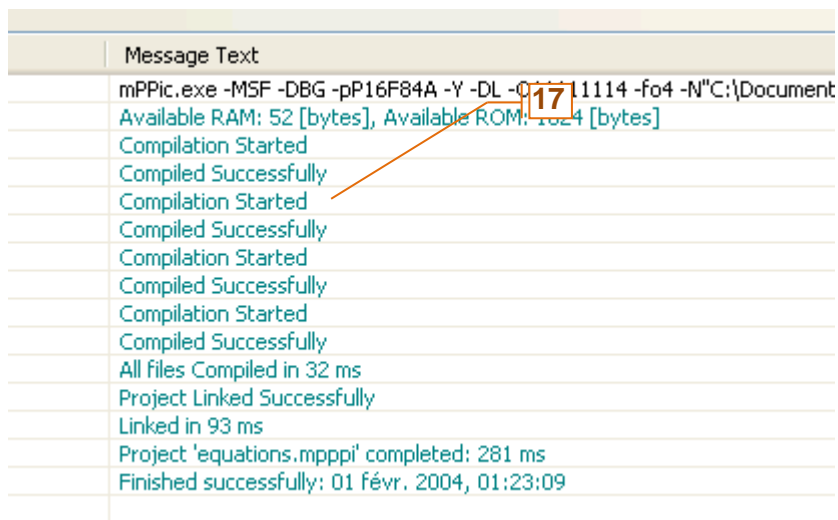
```
                S4:= (not A) and B or D;
```

```
                S5:= not (A xor B);
```

```
            end;
```

```
end.
```

17-Message «Compiled Successfully » dans l'onglet messages.



U1

16 OSC1/CLKIN RA0 17

15 OSC2/CLKOUT RA1 18

4 MCLR RA2 1

RA4/T0CKI RA3 2

3

RB0/INT 6

RB1 7

RB2 8

RB3 9

RB4 10

RB5 11

RB6 12

RB7 13

PIC16F84A

0 A

0 B

0 D

? S1

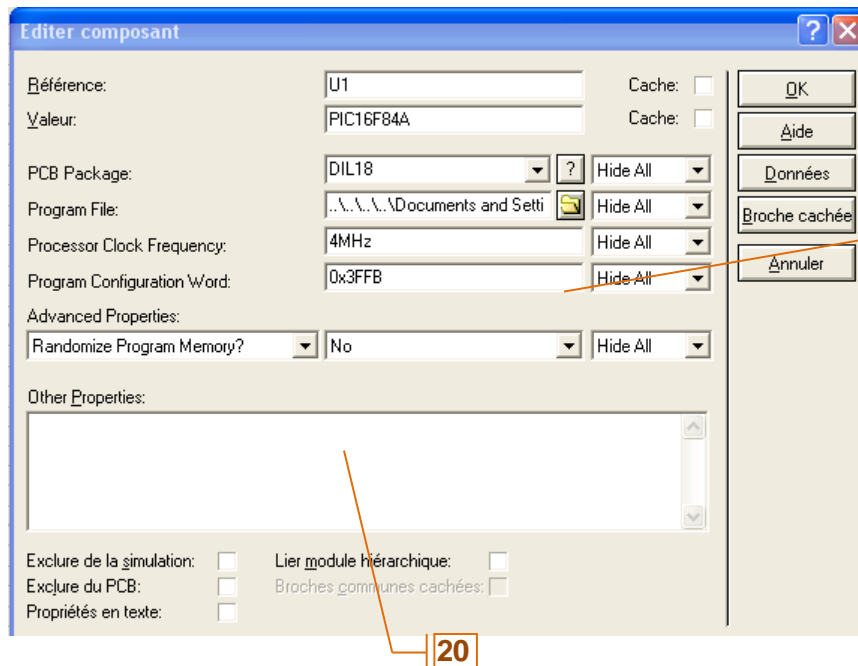
? S2

? S3

? S4

? S5

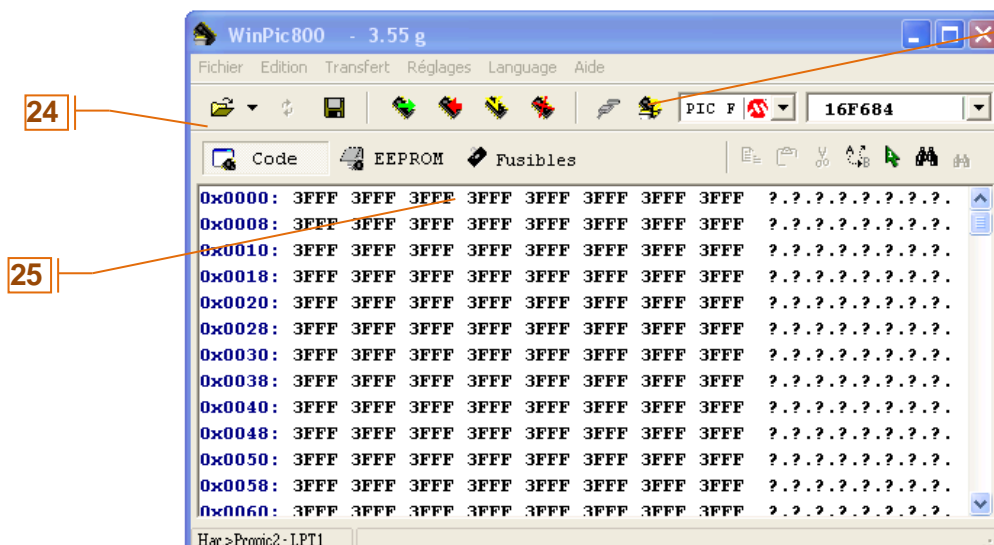
19- Dans la fenêtre « Edit Component » indiquer le nom et le chemin du fichier.hex



20- Régler l'horloge à 4MHz

21- Lancer la simulation et vérifier le fonctionnement des sorties.

22- Lancer le logiciel WinPic800.



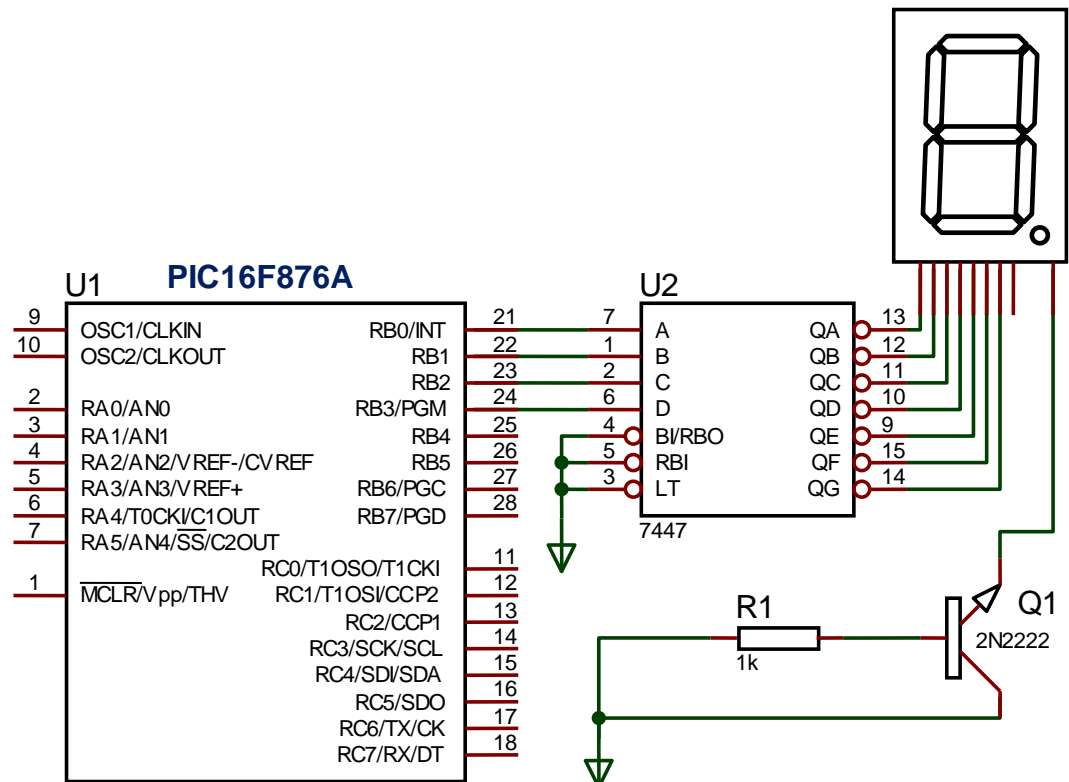
23- Cliquer sur l'icône de détection du type du microcontrôleur.

24- Dans le menu principal cliquer sur : Fichier ... Ouvrir ... equations.hex

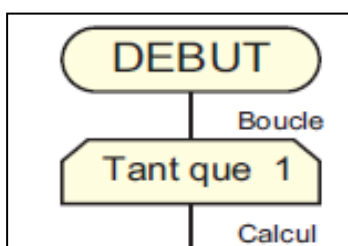
25- Programmer le microcontrôleur PIC16F84A

## Activité N° 2: Compteur à base de PIC16F876A

On souhaite réaliser un compteur modulo 10 avec le PIC 16F876A suivant ce montage :



Traduire l'algorithme suivant en un programme micropascal.



```
program ACTIVITE2;
```

```
Var
```

```
N:byte; // N est une variable de type octet
```

```
begin
```

Activité N° 3: Affichage multiplexé

On désire afficher un nombre formé de 3 chiffres exemple 932 en utilisant 3 afficheurs à 7 segments

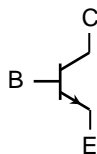
et un seul décodeur, pour cela on utilise un microcontrôleur 16F84A qui réalise le multiplexage de l'affichage. Le décodeur utilisé est le 7447 dont les sorties sont activées à niveau bas donc les afficheurs sont à anodes communes.

Les bornes communes des afficheurs « anodes » sont commandées à travers des transistors NPN de telle sorte que lorsque un transistor est saturé, la borne commune de l'afficheur est alors reliée au +Vcc donc l'afficheur correspondant fonctionne.

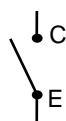
Le principe est de placer le nombre à afficher sur le décodeur puis commander le transistor correspondant pour l'afficher.

### Schéma du montage :

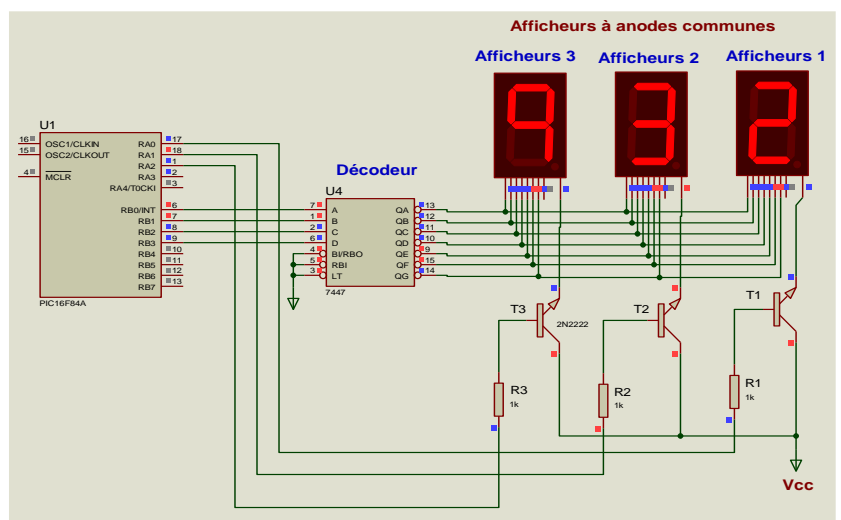
Les transistors utilisés pour la commande des afficheurs sont de type **NPN**.



- Si  $B=1$  le transistor est saturé



- Si  $B=0$  le transistor est bloqué



1° ) Compléter le tableau suivant :

Nombre à afficher	Transistor : bloqué ou saturé			Afficheur commandé : oui ou non			Temporisation
	T1	T2	T3	Afficheur 1	Afficheur 2	Afficheur 3	
	bloqué	bloqué	bloqué	non	non	non	1ms
2	saturé	bloqué	bloqué	oui	non	non	10ms
	bloqué	bloqué	bloqué	non	non	non	1ms
3	bloqué	saturé	bloqué	non	oui	non	10ms
	bloqué	bloqué	bloqué	non	non	non	1ms
9	bloqué	bloqué	saturé	non	non	oui	10ms

2° ) Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

program affichage\_multiplexe;

begin

trisb:=\$F0 ; // RB0,RB1,RB2,RB3 sorties

trisa:=\$18 ; // RA0,RA1,RA2 sorties,

RA3,RA4 entrées

porta:=0; // état initial du porta

While true do

begin

porta:=%000 ; // aucun afficheur

delay\_ms(1);

portb:=2; // placer le nombre de l'unité

porta:=%001; // Commander le 1<sup>er</sup> afficheur

porta:=%000; // aucun afficheur

delay\_ms(1);

portb:=3 ; // placer le nombre de dizaine

porta:=%010 ; // Commander le 2<sup>ème</sup> afficheur

delay\_ms(10) ;

porta:=%000; // aucun afficheur

delay\_ms(1);

portb:=9 ; // placer le nombre de centaine

porta:=%100; // Commander le 3<sup>ème</sup> afficheur

delay\_ms(10);

end;

end.

```
delay_ms(10);
```

#### Activité N° 4: Compteur modulo 1000 avec affichage multiplexé

Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program compteur_modulo1000_affichage_multiplexe;
```

```
var i:word;
```

```
var j:byte;
```

```
var unite,dizaine,centaine:byte;
```

```
begin
```

```
trisb:=$F0 ; // RB0,RB1,RB2,RB3 sont les entrées
```

```
trisa:=$18 ; // RA0,RA1,RA2 sont les sorties
```

```
porta:=0 ; // état initial du porta
```

```
While true do
```

```
begin
```

```
for i:=0 to 999 do
```

```
begin
```

```
unite:= i mod 10 ; // identifier le chiffre de l'unité
```

```
dizaine:= (i div 10) mod 10 ; // identifier le chiffre de dizaine
```

```
centaine:= i div 100 ; // identifier le chiffre de centaine
```

```
for j:=1 to 28 do
```

```
begin
```

Comment identifier les chiffres unité, dizaine, centaine et millier d'un nombre ?  
Exemple :N=9573

N=9573		
Unité de N	3	$N \bmod 10$
Dizaine de N	7	$(N \div 10) \bmod 10$
Centaine de N	5	$(N \div 100) \bmod 10$
Millier de N	9	$N \div 1000$



```
porta:=%000;

delay_ms(1);

portb:=unite;

porta:=%001;    // Commander le premier afficheur

delay_ms(10);

porta:=%000;

delay_ms(1);

portb:=dizaine;

porta:=%010;    // Commander le 2ème afficheur

delay_ms(10);

porta:=%000;

delay_ms(1);

portb:=centaine;

porta:=%100;

delay_ms(10);

end;

end;

end;

end.
```

ActivitéN° 5:signalisation lumineuse d'un robot (voir activité de découverte)

Compléter puis simuler le programme suivant :

```
program activite5;
```

```
var i:byte; // declaration d'une variable i de type octet
```

```
begin
```

```
    TRISB:= 0 ;    // Configuration du port B comme sortie
```

```
    TRISA:=$FF; // Configuration du port A comme entrée
```

```
    PORTB:=0; // initialisation du port B
```

```
    CMCON:=$07; // Désactivation du comparateur, PORTA numérique
```

```
While true do // Boucle infinie
```

```
begin
```

```
    if PORTA= 0 then PORTB:= 0; // Arrêt toutes les diodes sont  
éteintes
```

```
    if PORTA= 3 then PORTB:=$FF; // Charge de la batterie toutes  
sont allumées
```

```
    if PORTA= 1 then // Batterie à la limite de décharge  
clignotement des diodes
```

```
begin
```

```
    PORTB:=$ FF;
```

```
delay_ms(100) ;
```

```
PORTB:=$00 ;
```

```
delay_ms(100) ;
```

```
end;
```

```
if PORTA= 2 then      // fonctionnement normal chenillard
```

```
begin
```

```
PORTB:=%10000000;
```

```
for i:=1 to 8 do      // Boucle pour
```

```
begin
```

```
delay_ms(100);
```

```
PORTB:=PORTB shr 1 ;  // décalage
```

```
à droite de 1 bit du PORTB
```

```
end;
```

```
end;
```

```
end;
```

```
end.
```

Activité N° 5 bis : signalisation lumineuse d'un robot (Utilisation des procédures)

```
program Activite 5bis;
```

```
var
```

```
diodes: byte at portB;  // On déclare une variable diodes de type octet affecté au  
PORTB pour les sorties
```

```
entrees: byte at portA; // On déclare une variable entrées de type octet affecté au  
PORTA pour les entrées
```

```
{  procedure marche(const etat:byte);  // déclaration d'une procédure qui porte le nom  
marche
```

```
begin  // la procédure marche reçoit du programme principal une  
constante etat de type octet
```

```
    diodes:=etat;
```

```
end;
```

```
{  procedure chenillard();  // déclaration d'une procédure qui porte le nom  
chenillard  
var i:byte;  // On déclare une variable i de type octet  
begin  
    diodes:=%10000000;  // cette procédure allume les diodes l'une après  
l'autre (chenillard)
```

```
for i:=1 to 8 do
```

```
begin
```

```
    delay_ms(100);
```

```
    diodes:=diodes shr 1 ;
```

```
end;
```

```
end;
```

```
procedure clignotant();           // déclaration d'une procédure qui porte le nom
```

clignotant

```
begin
```

```
    marche($FF);delay_ms(100);    // tous les diodes sont allumées pendant 100ms
```

```
    marche($00);delay_ms(100);    // tous les diodes sont éteintes pendant 100ms
```

```
end;
```

```
begin
```

```
    trisb:=$00;TRISA:=$FF;        // Configuration du PORTB en sortie et
```

PORTA en entées

```
    cmcon:=$07;    // PORTA numérique désactivation du comparateurs
```

analogiques

```
    while true do    // boucle infinie
```

```
begin
```

```
if entrees=0 then marche(0); // Appel de la procédure marche qui reçoit la valeur
```

0 « diodes éteintes »

```
if entrees=1 then clignotant(); // Appel de la procédure clignotant
```

```
if entrees=2 then chenillard(); // Appel de la procédure chenillard
```

```
if entrees=3 then marche($FF); // Appel de la procédure marche qui reçoit la  
valeur 255 « diodes allumées »
```

```
end;
```

```
end.
```

### 😊 ActivitéN° 6 (Grafcet1)

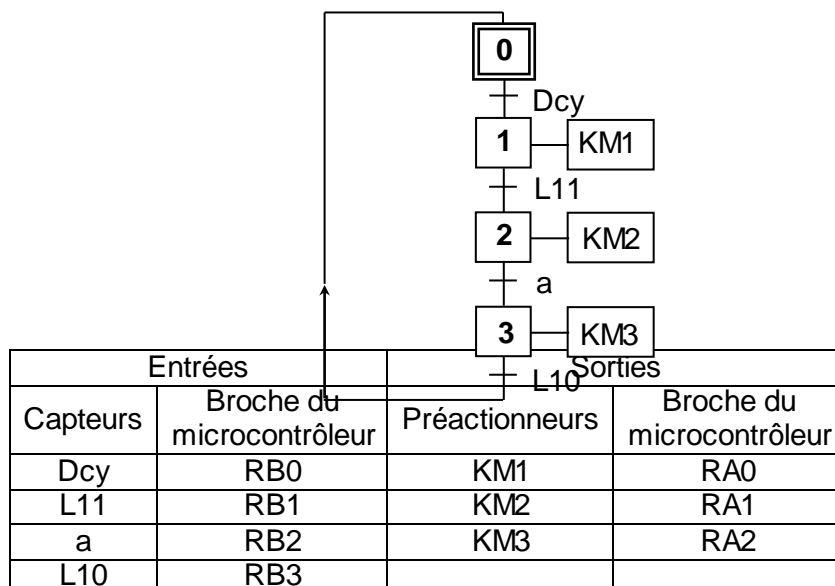
Objectif : L'objectif de cette activité est de traduire un grafcet d'un point de vue partie commande en un algorithme puis le traduire en programme en Mikropascal.

GRAFCET d'un point de vue P.C

On vous donne le tableau d'affectation des entrées /sorties

Pour le microcontrôleur PIC

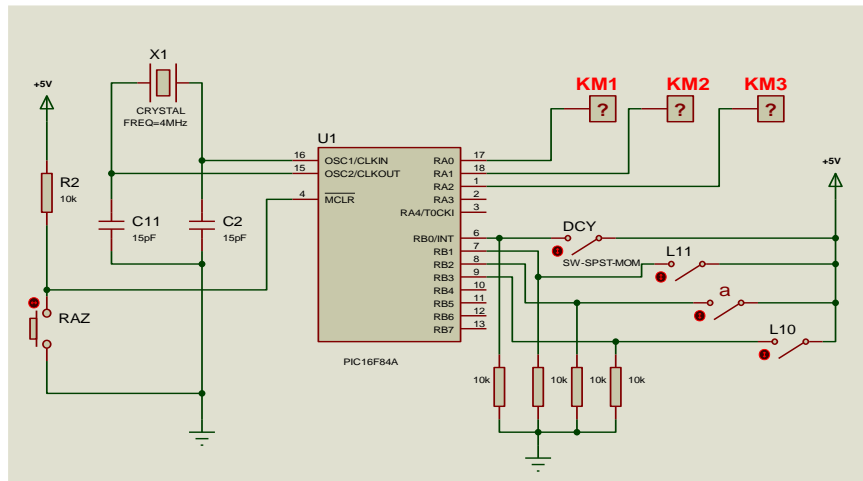
16F84A :



- ❖ Compléter l'algorithme.
- ❖ Compléter le programme en Mikropascal.

Algorithme	Programme
<pre> Algorithme grafcet_1; Variables :   Dcy :un bit affecté à RB0 ;   L11 :un bit affecté à RB1 ;   a   : un bit affecté à RB2 ;   L10 :un bit affecté à RB3 ;   KM1 :un bit affecté à RA0 ;   KM2 :un bit affecté à RA1 ;   KM3 :un bit affecté à RA2 ; X0, X1, X2, X3:bit ; début   trisa ← \$00; trisb ← \$FF;   KM1 ← 0 ; KM2 ← 0 ;KM3 ← 0 ;   X0 ← 1; X1 ← 0; X2 ← 0; X3 ← 0;   Tant que (vrai) faire   début     si (X0=1) ET (Dcy=1) alors       début         X0 ← 0; X1 ← 1;       fin;     si (X1=1) ET ( L11 =1) alors       début         X1 ← 0; X2 ← 1;       fin;     si (X2=1) ET ( a =1) alors       début         X2 ← 0; X3 ← 1;       fin;     si (X3=1) ET ( L10 =1) alors       début         X3 ← 0; X0 ← 1;       fin;   si (X1=1) alors KM1← 1 sinon KM1 ← 0   si (X2=1)-alors KM2 ← 1 sinon KM2 ← 0   si(X3=1); alors KM3← 1 sinon KM3 ← 0   finfaire; fin. </pre>	<pre> program grafcet_1; var   Dcy : sbit at RB0_bit ;   L11 : sbit at RB1_bit ;   a   : sbit at RB2_bit ;   L10 : sbit at RB3_bit ;   KM1 :sbit at RA0_bit ;   KM2 :sbit at PORTA.1 ;   KM3 :sbit at PORTA.2; X0, X1, X2, X3:bit; begin   trisa := \$00; trisb := \$FF;   KM1 := 0 ; KM2 := 0 ;KM3 := 0 ; // initialisation des sorties   X0 :=1; X1 := 0; X2 := 0; X3 := 0; // initialisation des étapes   while true do // boucle infinie   begin     if (X0=1) and (Dcy=1) then       begin         // désactivation: étape 0         X0 := 0; X1 := 1; // activation: étape1       end;     if (X1=1) and ( L11 =1) then       begin         X1 := 0; X2 :=1;       end;     if (X2=1) and ( a =1 ) then       begin         X2 := 0; X3 :=1;       end;     if (X3=1) and ( L10 =1 ) then       begin         X3 := 0; X0 :=1;       end;     if (X1=1) then KM1:= 1 else KM1:= 0; // sortie: KM1     if (X2=1) then KM2:= 1 else KM2:= 0; // sortie: KM2     if (X3=1) then KM3:= 1 else KM3:= 0; // sortie: KM3   end; end. </pre>

Ecrire le programme, le compiler et vérifier le fonctionnement :

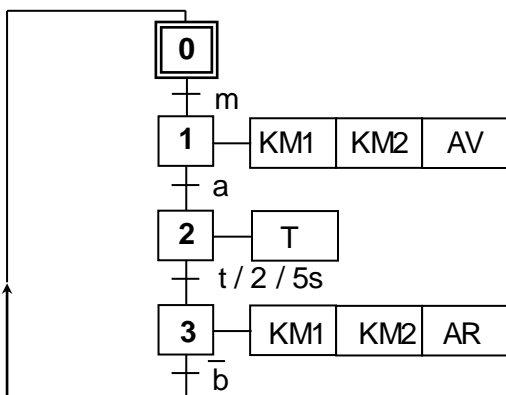


## 😊 Activité N° 7 (Grafcet2)

On vous donne le tableau d'affectation des entrées /sorties

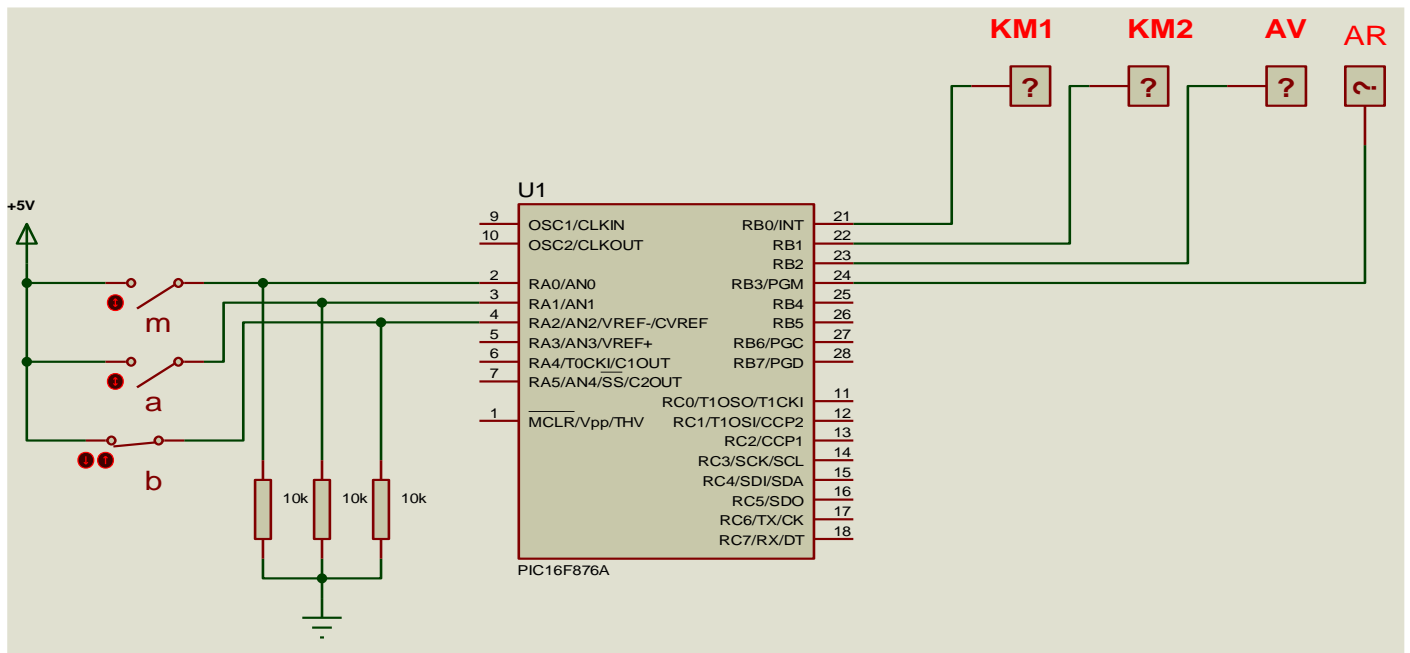
Pour le microcontrôleur

PIC 16F876A :



Entrées		Sorties	
Capteurs	Broche du microcontrôleur	Préactionneurs	Broche du microcontrôleur
m	RA0	KM1	RB0
a	RA1	KM2	RB1
b	RA2	AV	RB2
		AR	RB3



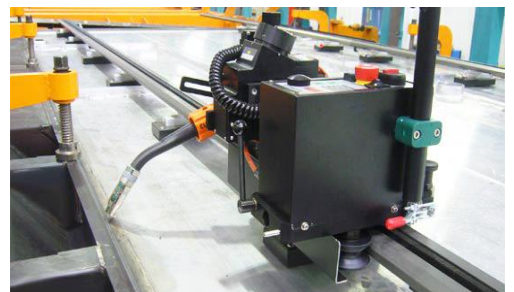


❖ Compléter le programme en Mikropascal.

Algorithme	Programme
<p>Algorithme grafcet_2;</p> <p>variables</p> <p>m: un bit du registre PORTA affecté à RA0 ;</p> <p>a: un bit du registre PORTA affecté à RA1 ;</p> <p>b: un bit du registre PORTA affecté à RA2 ;</p>	<pre> program grafcet_2; var m: sbit at RA0_bit ; a: sbit at RA1_bit ; b: sbit at RA2_bit ; KM1: sbit at PORTB_0 ; KM2: sbit at PORTB_1 ; AV: sbit at PORTB_2 ; AR: sbit at PORTB_3 ; </pre>

😊 **Activité 8:** GRAFCET3 (séquences multiples) :

Chariot de soudage mobile sur rail fixe

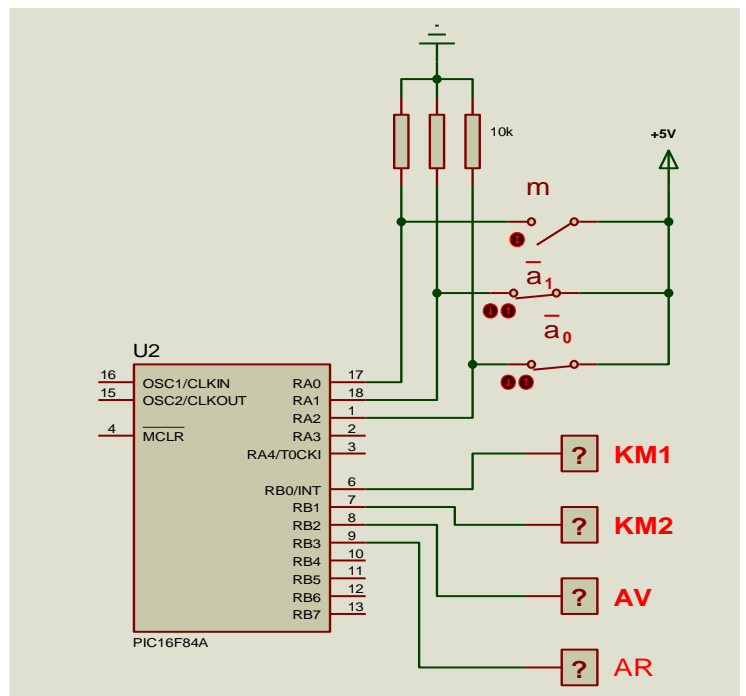
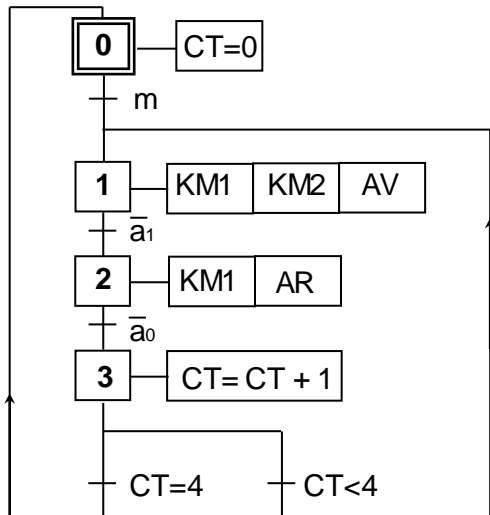


## Présentation :

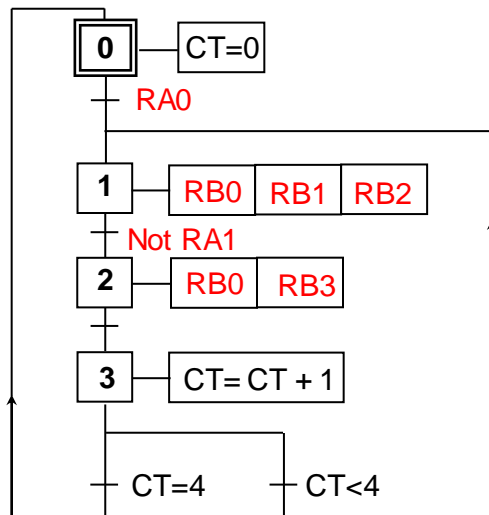
Un appui sur le bouton poussoir «m» provoque le déplacement du chariot et active la torche de soudage.

Le chariot effectue 4 cycles de va et vient puis s'arrête.

Entrées			Sorties		
Désignation		Broche du microcontrôleur	Désignation	Préactionneurs	Broche du microcontrôleur
Mise en marche	m	RA0	Translation du chariot	KM1	RB0
Fin de course droite	a <sub>1</sub>	RA1	Torche de soudage	KM2	RB1
Fin de course gauche	a <sub>0</sub>	RA2	Sens avant pour la translation du chariot	AV	RB2
			Sens arrière pour la translation du chariot	AR	RB3



## Grafcet codé PIC



Traduire le GRAFCET d'un point de  
vue partie commande ci-  
dessous en un GRAFCET codé  
microcontrôleur puis le traduire  
en programme en Mikropascal.

```

program Grafcet_soudure;
  var
    m : sbit at RA0_bit ;
    a0: sbit at RA2_bit;
    a1: sbit at RA1_bit ;
  
```

```

KM1:sbit at RB0_bit ;
KM2:sbit at RB1_bit ;
AV:sbit at RB2_bit ;
AR: sbit at RB0_bit ;
X0,X1,X2,X3:bit;
CT:byte;

begin

    trisa := $1F; trisb := $00;

    KM1:= 0 ;KM2 := 0 ;AV := 0 ;AR := 0 ;

    X0 :=1; X1 := 0; X2 := 0; X3 := 0;

    while true do

        begin

            if (X0=1) and (m=1) then

                begin

                    X0 := 0; X1 := 1;

                end;

            if (X1=1) and ( a1 = 0) then

                begin

                    X1 := 0; X2 :=1;

                end;

            if (X2=1) and ( a0 = 0 ) then

                begin

                    X2 := 0; X3 :=1;CT:=CT +1 ;

```

```

end;

if (X3=1) and (CT < 4 ) then

begin

X3 := 0; X1 :=1;

end;

if (X3=1) and (CT = 4 ) then

begin

X3 := 0; X0 :=1;CT:=0;

end;

if (X1=1) or (X2=1) then KM1:= 1 else KM1:= 0;
if (X1=1) then KM2:= 1 else KM2:= 0;

if (X1=1) then AV:= 1 else AV:= 0;
if (X2=1) then AR:= 1 else AR:= 0;

end;

end.

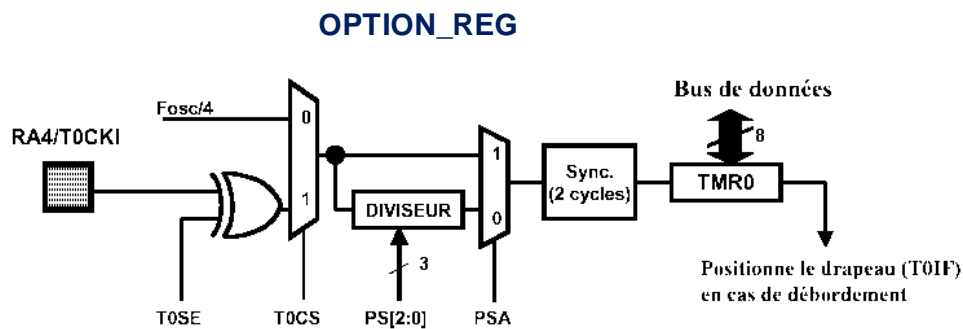
```

## LE TIMER TMR0

Le registre TMR0 est un compteur programmable de 8 bits (de 0 à 255).

La configuration du TMR0 est assurée par le registre OPTION « [OPTION\\_REG](#) »

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBP	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0



PS2	PS1	PS0	Diviseur
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

Tableau 1

### Schéma synoptique du registre OPTION

Le TMR0 est incrémenté en permanence soit par :

- L'horloge interne ( $f_{osc}/4$ ) « mode TIMER »
- L'horloge externe appliquée à la broche RA4 du portA « mode compteur »

Le choix de l'horloge se fait à l'aide du bit 5 du registre OPTION\_REG « TOCS »

- TOCS = 0 Horloge interne « mode TIMER »
- TOCS = 1 Horloge externe « mode COMPTEUR »

Dans le cas de l'horloge externe, le bit 4 « TOSE » du registre OPTION\_REG permet de choisir le front sur lequel le TIMER0 s'incrémente :

- TOSE = 0 incrémentation sur fronts montants
- TOSE = 1 incrémentation sur fronts descendants

Quelque soit l'horloge choisie, on peut la faire passer dans un diviseur de fréquence programmable (prescaler) dont le rapport est fixé par les bits **PS0,PS1 et PS2** du registre **OPTION\_REG** « voir tableau 1 »

L'affectation ou non du prédiviseur se fait à l'aide du bit 3 « **PSA** » du registre **OPTION\_REG**

- **PSA =0** on utilise le prédiviseur.
- **PSA =1** pas de prédiviseur.

**Bit 6 :INTEDG** « **INTerrupt Edge** » : dans le cas où on utilise l'interruption externe avec **RB0**

- Si **INTEDG = 1**, on a interruption si le niveau sur **RB0** passe de 0 vers 1. « front montant »
- Si **INTEDG = 0**, l'interruption s'effectuera lors de la transition de 1 vers 0. « front descendant »

---

**Bit 7 : RBPU**: Quand ce bit est mis à 0, une résistance de rappel au +5 volt est placée sur chaque broche du **PORTB**

**N.B** À l'issue d'un *Reset*, le registre **OPTION\_REG** = 11111111

😊 **Activité 9** : On désire réaliser un compteur modulo 10 en utilisant le timer **TMR0**. En mode compteur « entrée de comptage sur **RA4** » ↑ avec un coefficient de prédivision = 1 (sortie sur le port **B**)



1° ) Configurer le registre « OPTION\_REG »

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPV	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
1	1	1	0	1	0	0	0

2° ) Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

program

  activite\_TMR0\_compteur\_10;

begin

    TRISB:=\$F0;

    TRISA:=\$1F;

    OPTION\_reg:= %11101000;

    TMR0:=0;

  while true do

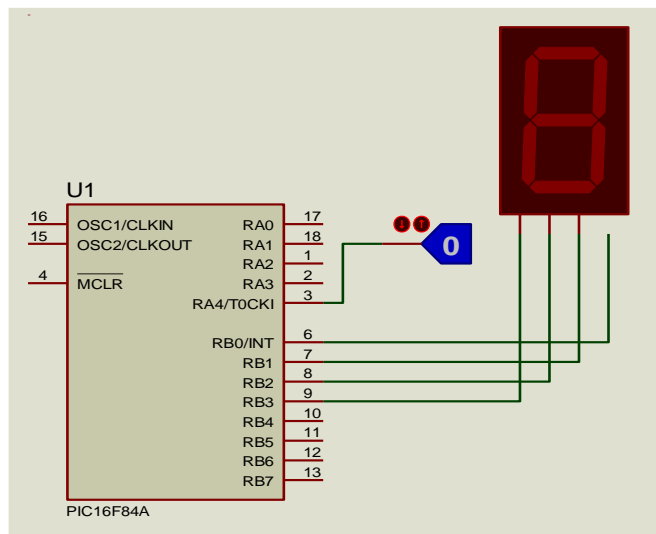
  begin

    portb:=TMR0;

    if TMR0=10 then TMR0:=0;

  end;

end.



**Activité 10:** On désire réaliser un compteur modulo 8 en utilisant le timer TMR0

Configurer ce même registre pour un prédiviseur = 2 « entrée de comptage sur RA4 »  
front ↓

1° ) Configurer le registre « OPTION\_REG »

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
1	1	1	1	0	0	0	0

2° ) Compléter puis saisir le programme ci-contre puis simuler et vérifie le  
fonctionnement :

```
program activite2_tmr0_compteur_8;
```

```
begin
```

```
    TRISB:=$F0;
```

```
    TRISA:=$1F;
```

```
    OPTION_reg:= %11110000;
```

```
    TMR0:=0;
```

```
    while true do
```

```
        begin
```

```
            portb:= TMR0;
```

```
            if TMR0=8 then TMR0:=0;
```

```
        end;
```

```
end.
```

😊 **Activité 11** : On désire réaliser un décompteur modulo10 en utilisant le timer TMR0 en mode compteur « entrée de comptage sur RA4 »↑ avec un coefficient de prédivison =1 ( sortie sur le port B)

1° ) Configurer le registre « OPTION\_REG »

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPUR	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
1	1	1	0	1	0	0	0

2° ) Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program activite_3_tmr0_decompteur_10;
```

```
begin
```

```
TRISB:=$F0;
```

```
TRISA:=$1F;
```

```
OPTION_reg:= %11101000;
```

```
TMR0:=0;
```

```
while true do
```

```
begin
```

```
portb:= 9 - TMR0;
```

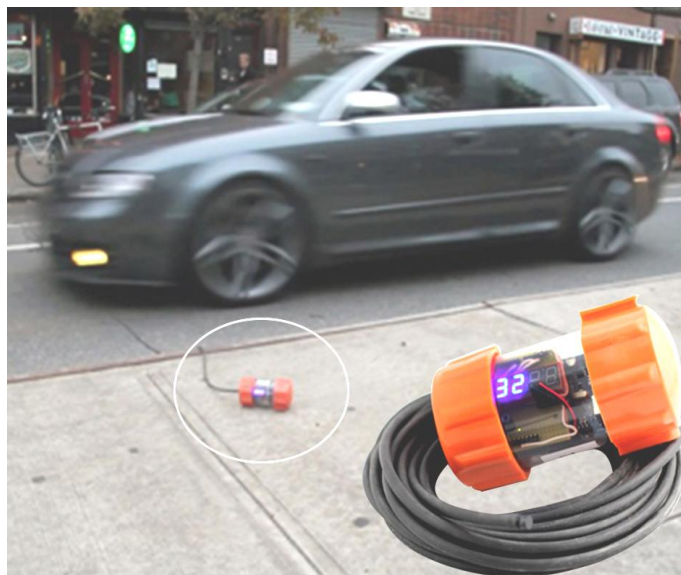
```
if TMR0=10 then TMR0:=0;
```

end;

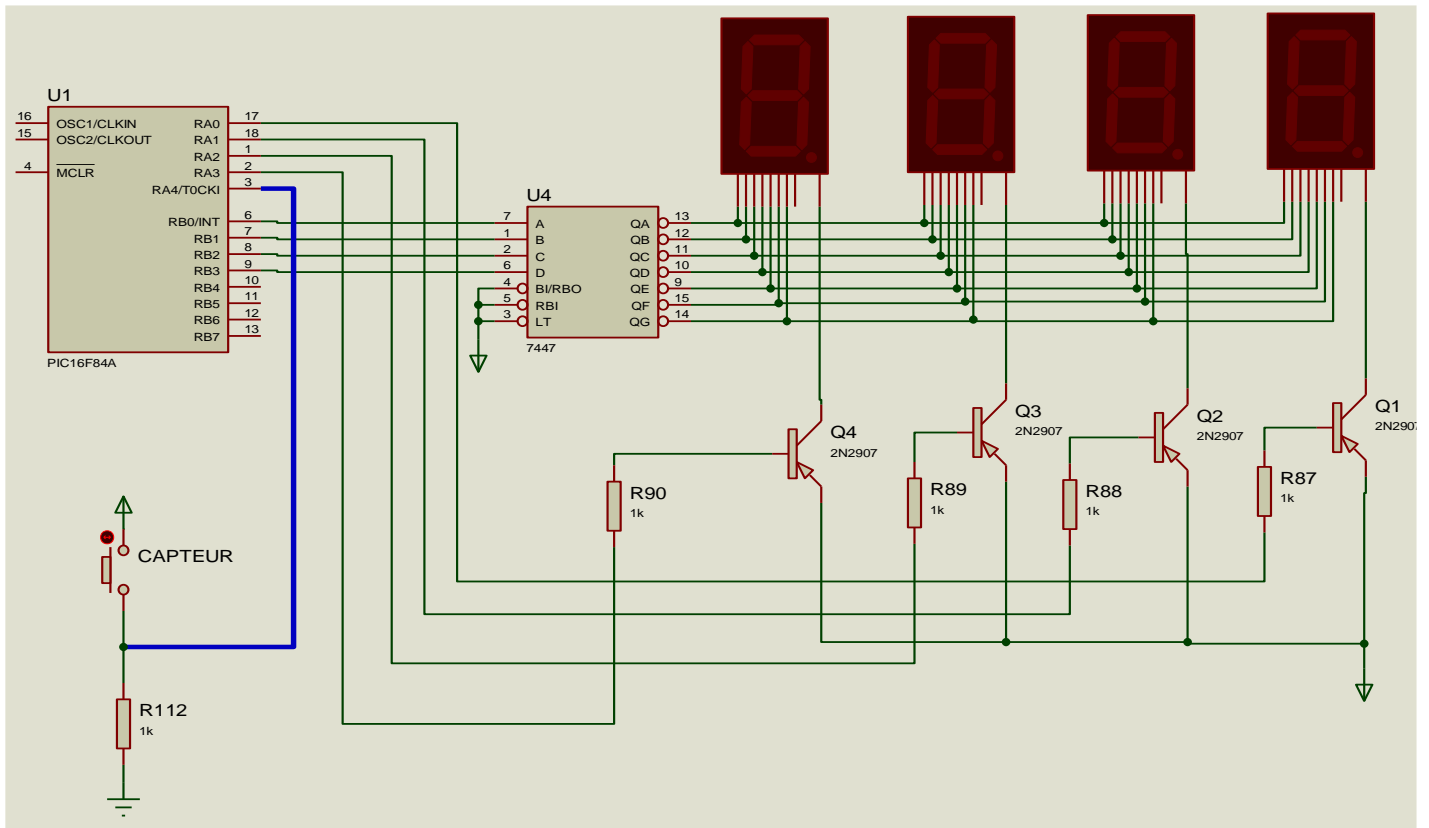
end.

### ☺ Activité 12 : Comptage de véhicules sur une route:

Ce système permet l'enregistrement du nombre de véhicules qui passent sur une route durant un temps donné. Le système est constitué d'un tuyau sensible à la pression, relié à une carte à base de microcontrôleur PIC 16F628A et un afficheur 7 segments 4 digits, le tout contenu dans un boîtier résistant aux intempéries en plastique transparent. Pour l'utiliser, il faut déployer le tuyau à travers la route, en le fixant avec une robuste bande adhésive puis, laisser les voitures rouler dessus. Chaque jeu de deux essieux (c'est-à-dire toutes les deux impulsions) est enregistré comme un véhicule.



## Schéma de montage



Les transistors utilisés pour la commande des afficheurs sont de type **PNP**.

- Si  $B=0$  le transistor est saturé



- Si  $B=1$  le transistor est bloqué



1° ) A chaque passage d'un véhicule, le capteur génère deux impulsions, justifier le choix de l'entrée RA4 du microcontrôleur PIC 16F84A pour relier ce capteur.

On a besoin d'une horloge externe pour compter le nombre d'impulsions : c'est la broche RA4 et on va utiliser le timer TMR0 en mode compteur.

2° ) Déterminer la valeur du registre OPTION\_REG

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBPUR	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
1	1	1	0	0	0	0	0

3° ) Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program activite6_voitures_comptage;
```

```
var
```

```
  i: word;
```

```
  a, uni, dix, cent, mil : byte;
```

```
const
```

```
aucun_afficheur : byte = %1111;
```

```
afficheur1 : byte = % 1110 ;           // commande de l'afficheur 1
```

```
afficheur2 : byte = %1101 ;           // commande de l'afficheur 2
```

```
afficheur3 : byte = %1011;           // commande de l'afficheur 3
```

```

afficheur4 : byte = % 0111;           // commande de l'afficheur 4
begin

trisb:=$F0;           // de RB0 à RB3 sorties ,de RB4 à RB7 entrées
trisa:=$10;           // de RA0 à RA3 sorties , RA4 entrée
i:=0;a:=0;           // Initialisation des variables i et a à la valeur 0
TMR0:=0;           // initialisation du timer 0 à la valeur 0
OPTION_REG:= %1100000; // TMR0 en mode compteur qui s'incrémente toutes les
deux impulsions.
while true do
begin
    if TMR0 > 200 then
        begin
            TMR0:=TMR0-200;           // pour étendre le comptage à des valeurs
supérieures à 255
            a:=a+1;
            end;
        i:=TMR0 + a*200 ;
        uni := i mod 10;           // Identifier le chiffre de l'unité de la variable i
        dix := (i div 10) mod 10; // Identifier le chiffre de dizaine de la variable i
        cent := (i div 100) mod 10; // Identifier le chiffre de centaine de la variable i
        mil := (i div 1000); // Identifier le chiffre de millier de la variable i
        porta:=aucun_afficheur;
        delay_ms(1);
        portb:=uni;
        porta:=afficheur1;
        delay_ms(10);
        porta:=aucun_afficheur ;
    end
end

```

```

    delay_ms(1);
    portb:=dix;
    porta:=afficheur2;
    delay_ms(10);
    porta:=aucun_afficheur ; // affichage multiplexé puisqu'on dispose d'un seul
décodeur
    delay_ms(1);
    portb:=cent;
    porta:=afficheur3;
    delay_ms(10);
    porta:=aucun_afficheur ;
    delay_ms(1);
    portb:=mil;
    porta:=afficheur4;
    delay_ms(10);
end;
end.

```

## NOTION D'INTERRUPTION

### 1° ) Définition :

Une interruption est un événement qui provoque l'arrêt d'un programme en cours d'exécution pour aller exécuter un autre programme appelé programme d'interruption (ou routine).

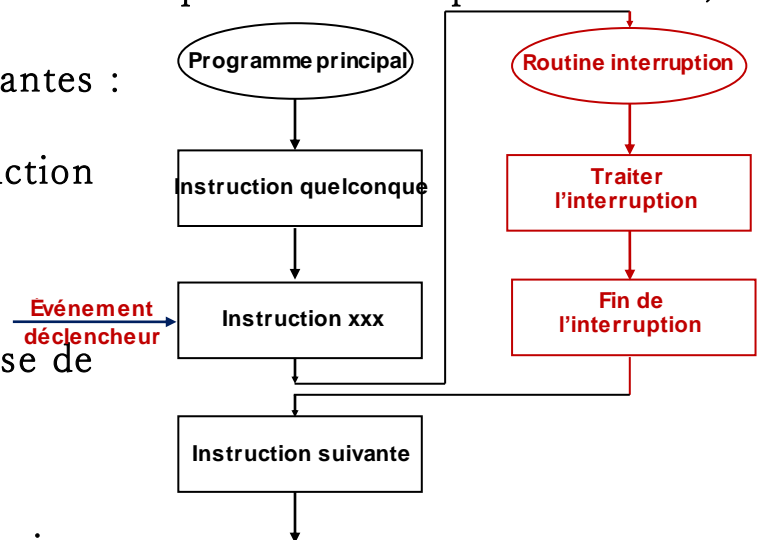
A la fin du programme d'interruption, le microcontrôleur reprend le programme principal à l'endroit où il s'est arrêté.

### 2° ) Déroulement d'une interruption :



Le programme se déroule normalement lorsque une interruption survient, le programme exécute les tâches suivantes :

- Le programme achève l'instruction en cours de traitement
- Le programme saute à l'adresse de traitement de l'interruption
- Le programme traite l'interruption
- Le programme revient à l'instruction qui suit la dernière exécutée dans le programme principal



### 3° ) Les sources et types d'interruptions :

Le nombre de sources d'interruptions dépend du microcontrôleur utilisé

#### Exemples:

Microcontrôleur	16F84A	16F628A	16F88	16F876A	16F877A
Sources d'interruption	4	10	12	14	15

- On distingue deux types d'interruptions :
  - Internes
  - externes

## Exemple: PIC 16F84A

Le PIC 16F84A dispose que de 4 sources d'interruptions, les événements susceptibles de déclencher une interruption sont les suivants :

- TMR0 : Débordement du timer0 (TMR0). Une fois que le contenu du TMR0 passe de 'FF' à '00', 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 broche RB0, encore appelée INTerrupt pin, étant configurée en entrée, le niveau qui lui 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.

### 4° ) Registre de configuration des interruptions (INTCON) :

Le registre INTCON (INTerrupt CONTroller) est le registre principal de contrôle et de gestion des interruptions.

Le registre INTCON est parfois différent d'un PIC à un autre il est impératif de revenir au document constructeur pour chaque type de microcontrôleur.

Registre INTCON pour PIC16F84A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Registre INTCON

**GIE:** « *Global Interrupt Enable* » mis à 1 autorise toutes les interruptions non masquées par leur bit individuel.

**EEIE:** « *EEPROM write completed Interrupt Enable* » : autorise les interruptions de fin d'écriture dans l'EEPROM.

**T0IE:** « *Timer0 Interrupt Enable* » : mis à 1 autorise les interruptions dues au débordement du *timer* 0.

**INTE:** « *INTerrupt Enablé* » : mis à 1, autorise les interruptions sur RB0/INT. L'interruption a lieu sur le front montant de l'impulsion si le bit INTEG (*INTerrupt Edge*) du registre OPTION est à 1 ; elle a lieu sur le front descendant si ce bit est à 0.

**RBIE:** « *RB Interrupt Enable* » : mis à 1, autorise les interruptions sur RB4 à RB7.

**T0IF:** « *Timer0 Interrupt Flag* » : est mis à 1 en cas de débordement du *timer* 0.

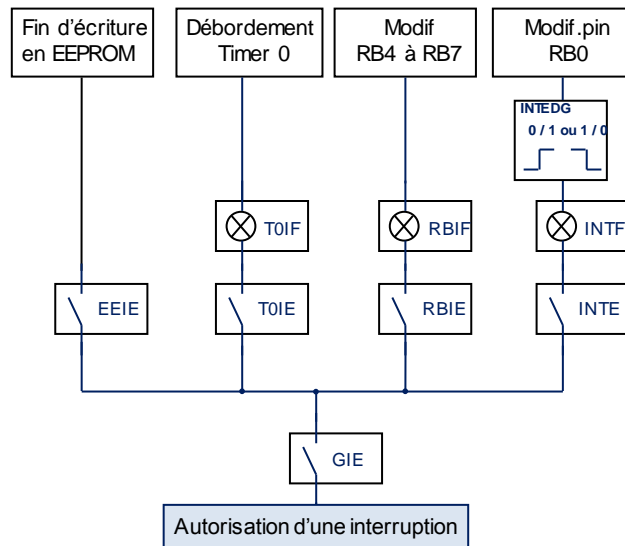
**INTF:** « *INTerrupt Flag* » : est mis à 1 si une interruption est générée sur RB0/INT.

**RBIF:** « *RB Interrupt Flag* », est mis à 1 lors d'un changement d'état sur une des lignes RB4 à RB7.

Chaque indicateur de changement d'état doit être remis à 0 par le logiciel dans le programme de traitement de l'interruption.

**NB :** Lors d'un *Reset*, tous les bits du registre INTCON sauf RBIF sont mis à 0. RBIF garde son état précédent.

Le schéma synoptique du registre INTCON



On s'intéresse uniquement aux deux types d'interruptions

☞ À l'interruption sur la broche RB0

☞ À l'interruption RB4 à RB7

5° ) Mise en œuvre d'une routine d'interruption en mikropascal

En mikropascal, le sous programme d'interruption est déclaré en tant que **procédure** avec le nom spécial « Interrupt » **procédure interrupt** .L'appel d'une procédure d'interruption ne dépend pas du programme principal, mais elle l'interrompt pendant son exécution.

Dans une procédure d'interruption on ne peut pas appeler une autre procédure.

procedure interrupt ;

Begin

Instruction 1;

.....;

Instruction n;

End;

### 5-1 Interruption RB0/INT :

☺ **Activité:** Configurer le registre INTCON en vue d'activer l'interruption externe sur RB0 (front montant)

AVANT L'INTERRUPTION : *Au cours de déroulement normal du programme principal :*

*Le bit7 (GIE=1)*

*Le bit4 (INTE=1)*

*Le bit (INTF =0 )*

INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Valeur INTCON
	1	0	0	1	0	0	0	0	(90) <sub>16</sub>

AU COURS DE L'INTERRUPTION : *Le microcontrôleur exécute la routine d'interruption :*

*Le bit7 (GIE=0)*

*Le bit4 (INTE=1)*

*Le bit (INTF=1)*

INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Valeur INTCON
	0	0	0	1	0	0	1	0	(12) <sub>16</sub>

Il faut impérativement mettre à 1 le bit GIE et mettre à zéro l'indicateur INTF à la fin du sous-programme d'interruption pour pouvoir revenir au programme principal et autoriser de nouvelles interruptions sur la broche RB0/INT. A la sortie de la routine d'interruption on écrit la valeur \$ 90 dans le registre INTCON

INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Valeur INTCON
	1	0	0	1	0	0	0	0	(90) <sub>16</sub>

CONCLUSION :

Déroulement normal du programme principal	Au cours d'une interruption	A la fin de l'interruption
INTCON := 0x90	INTCON := 0x12	INTCON := 0x90

Il faut toujours remettre le registre INTCON à 0x90 à la fin de la routine d'interruption pour autoriser une autre interruption ou bien mettre les bits : `INTCON.GIE :=1` et `INTCON.INTF :=0`

N.B si on veut produire l'interruption sur le front descendant d'une impulsion appliquée sur RB0 on doit

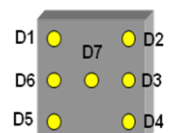
Mettre un 0 au bit `INTEDG` du registre `OPTION_REG`

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBP	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Si `INTEDG = 1`, l'interruption s'effectuera si le niveau sur RB0 passe de 0 vers 1. (front montant de RB0)

Si `INTEDG = 0`, l'interruption s'effectuera si RB0 passe de 1 vers 0. (front descendant de RB0)

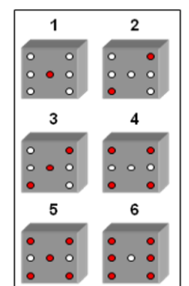
À l'issue d'un *Reset*, tous les bits du registre `OPTION` sont à 1.  
activité N° 1 (interruption avec RB0) : Dé électronique



➤Présentation :

Le dé électronique est constitué :

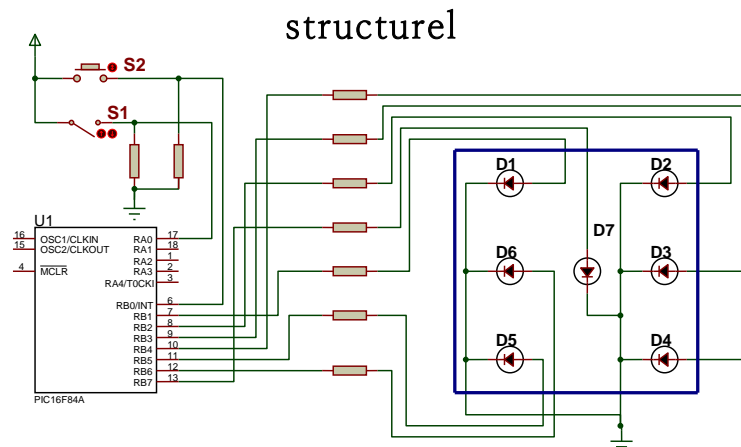
- D'une platine comportant 7 diodes leds D1, D2, D3, D4, D5, D6 et D7. (PORTB)
- Un étage de commande comportant un interrupteur S1



pour la marche /arrêt et un bouton poussoir S2 pour jouer

- Une carte électronique à base d'un microcontrôleur

➤ Schéma



➤ Description du fonctionnement :

Ce circuit génère un nombre des LEDs allumées de 1 à 6 de manière aléatoire (comme un dé à 6 faces).

Lorsqu'on met l'interrupteur S1 à 1, on aura une boucle qui incrémente le registre port B alimentant les 7 diodes LEDs donnant une valeur aléatoire (1-2-3-4-5-6-1-2 ...) suivant une base de temps de  $10\mu s$ .

Un appui sur le bouton poussoir S2 provoque une interruption (RB0/INT) : le contenu du registre port B reste à son état pendant environ 2 secondes indiquant un nombre de 1 à 6 affiché par les diodes LEDs.

➤ Configuration du registre INTCON :

INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Valeur INTCON
	1	0	0	1	0	0	0	0	(90) <sub>16</sub>

- Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program D_electronique;
```

```
{ Procedure interrupt;
begin
    delay_ms(2000);
```

```
intcon:=$90; //Mise à zéro du bit "INTF" du registre INTCON
```

```
end;
```

```
begin
```

```
while true do
begin
if porta.0=0 then portb:=%00000000 else
begin
portb:=%10000000; delay_us(10); //Allumer D7
portb:=%00100100; delay_us(10); //D2 et D5
portb:=%10100100; delay_us(10);
portb:=%00110110; delay_us(10);
portb:=%10110110; delay_us(10);
portb:=%01111110; delay_us(10);
end;
end;
end.
```

```
intcon:=$90; // Activation de l'interruption externe RB0/INT
```

```
Trisb:=$01; // Configurer le port B en sortie sauf RB0 entrée
```

```
Trisa:=$FF ; // Configurer le port A en entrées
```

```
portb:=0; // Initialiser le portB
```

## 5-2 Interruption RB4 à RB7 :

L'interruption est provoquée par un changement d'état sur au moins d'une des entrées RB4 à RB7

du port B. Les broches (RB4 à RB7) configurées en entrées sont comparées



périodiquement à l'ancienne valeur mémorisée par la dernière lecture du port B pour gérer l'interruption.

Lorsque le mécanisme de l'interruption RB4 à RB7 se déclenche le microcontrôleur verrouille le bit indicateur RBIF à 1, une opération de lecture sur le port B est nécessaire pour déverrouiller l'accès au bit RBIF afin de pouvoir le remettre à 0.

INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Valeur INTCON
	1	0	0	0	1	0	0	0	(88) <sub>16</sub>

😊 **Activité :** Configurer le registre INTCON en vue d'activer l'interruption externe sur RB4 à RB7

## Activité N° 2 (interruption avec RBI ) : Dé électronique

Refaire le même exemple mais en utilisant l'interruption sur RB4 à RB7 « PIC 16F876A »

Le dé électronique est constitué :

- D'une platine comportant 7 diodes leds D1, D2, D3, D4, D5, D6 et D7.(PORTC)
  - Un étage de commande comportant un interrupteur S0 pour la marche /arrêt et des boutons poussoirs S1, S2, S3 et S4 pour jouer.
- Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program de_electronique_interruption_RB;

```

```
    var etat:byte;

```

```
Procedure interrupt;

```

```
Begin

```

```
etat:=PORTB; //lecture du portB pour déverrouiller
```

l'accès au bit RBIF

```
delay_ms(2000);
```

```
intcon:=$88; //Mise à zéro du bit "INTF" du
```

registre INTCON

```
// ou intcon.rbif:=0 ; intcon.GIE:=1
```

```
end;
```

```
begin
```

```
intcon:=$ 88 ; //Activation de l'interruption externe RBI
```

```
Trisb:=$FF ; //Configurer le port B en entrées
```

```
TRISC:=0; // Configurer le portC en sortie
```

```
portc:=0; //Initialiser le portC
```

```
while true do begin
```

```
if portb.1=0 then portc:=%00000000 else
```

```
begin
```

```
portc:=%10000000; delay_us(10); // Allumer la led D7
```

```
portc:=%00100100; delay_us(10); // Allumer la led D2 et D5
```

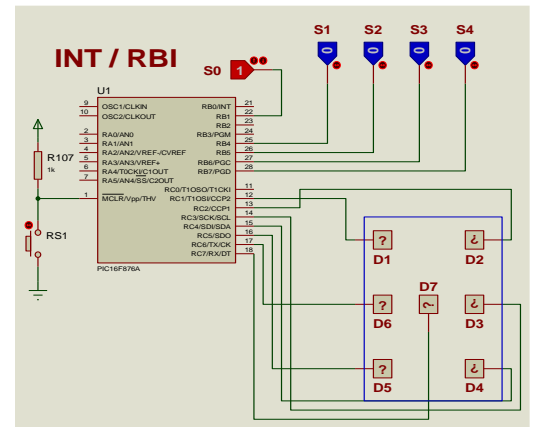
```
portc:=% 10100100 ; delay_us(10); // Allumer la led D2 , D5 et D7
```

```
portc:=% 00110110 ; delay_us(10); // Allumer la led D1 , D2 , D4 et D5
```

```
portc:=%10110110; delay_us(10); // Allumer la led D1 , D2 , D4 , D5 et D7
```

```
portc:=%01111110; delay_us(10); // Allumer la led D1, D2, D3, D4, D5 et D6
```

```
end;
```



end;  
end.

### Activité N° 3 (interruption avec RBI) : Coffre-fort électronique

Ce coffre-fort électronique sert à protéger des objets de valeur du vol. Pour l'ouvrir l'utilisateur doit introduire une séquence de 6 chiffres dans l'ordre via une roue codeuse par exemple : 5-4-7-10-9-12



Un bouton poussoir situé permet d'initialiser la séquence d'ouverture à partir du premier chiffre du code.

A chaque changement de la position de la roue, le microcontrôleur de la carte de commande du coffre enregistre les positions intermédiaires.

Exemple : Pour le code 5-4-7-10-9-12 la séquence correcte qui permet d'ouvrir la porte du coffre est :

5 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 9 → 10 → 11 → 12

1- On souhaite lire les 4 bits fournis par la roue codeuse à chaque changement d'état.

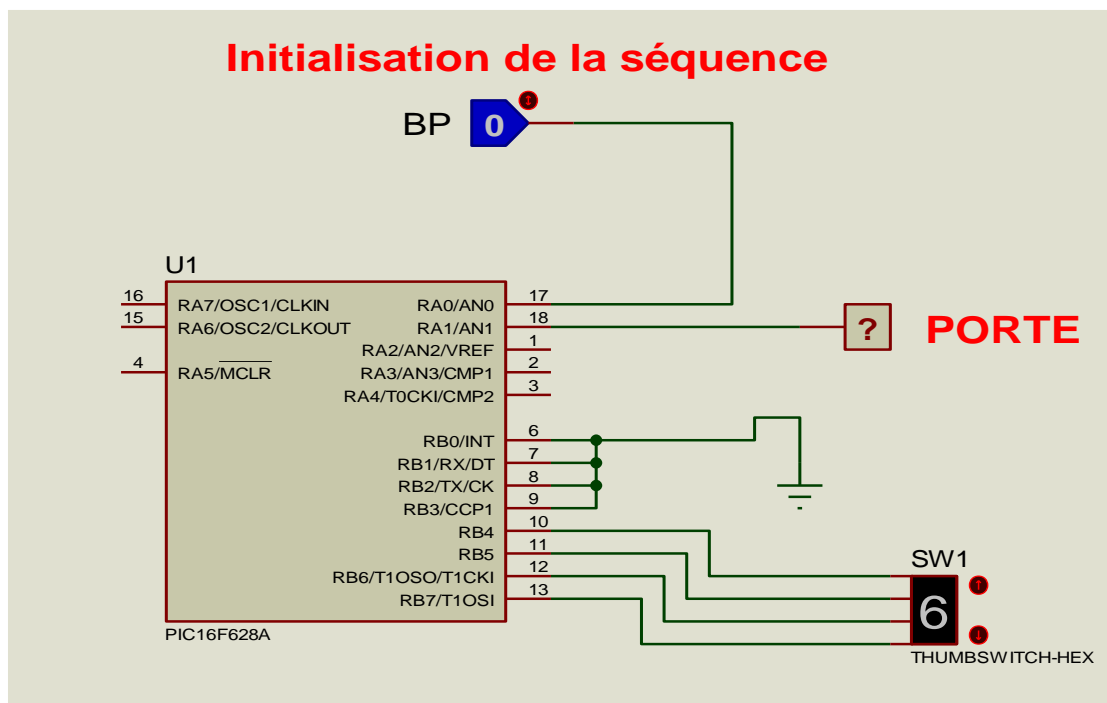
*Quelle est la technique la plus adaptée qui permet de lire cet événement quel que soit l'état du programme en cours d'exécution ?*

C'est l'interruption par le changement de l'un des états RB4, RB5, RB6 et RB7

2- Le microcontrôleur utilisé étant un PIC16F628A : donner la valeur du registre INTCON :

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
1	0	0	0	1	0	0	0

3- Saisir le programme ci-contre puis simuler et vérifier le fonctionnement :



program coffre\_fort ;

```

const
  code_acces:array[12]of byte=(5,4,5,6,7,8,9,10,9,10,11,12);
var
  code_saisi:array[12]of byte;
  i,j:byte;
  etat:byte;
  RAZ:sbit at PORTA.0;
  porte:sbit at porta.1;
  ok:bit;
  {
    Procedure interrupt;
    begin
      etat:= PORTB; // lecture du PORTB
      etat:=etat SHR 4 ; // Décalage à droite des 4 bits lus puisque la roue codeuse est
      branchée sur
                        RB4,RB5,RB6 et RB7

      code_saisi[j]:=etat ; // la colonne j prend la valeur de cet etat
      j:=j+1; // on incrémente à chaque fois j pour remplir les douzes colonnes du tableau
      avec 12 chiffres

      if j>11 then j:=0;

      INTCON.RBIF:=0; //Remise à zéro de l'indicateur RBIF

      INTCON.GIE:=1; // Réactivation globale des interruptions
    end;
    begin
      CMCON :=$07; // PortA numérique
      j:=0;
  }

```

```

TRISA.1:=0; // La broche RA1 est une sortie

porte:=0; // Initialisation de la variable porte à 0

INTCON:=%10001000; // ACTIVATION DE L'INTERRUPTION AVEC RBI

WHILE TRUE DO
begin
if RAZ =1 then
begin
code_saisi[0]:=portb shr 4 ; // Initialisation la séquence d'ouverture à partir du premier
chiffre du code

j:=1;

end;

ok:=1;

for i:=0 to 11 do

begin

// test de la sequence saisies ; on teste les douzes numéros du code d'accée et les
douzes numéros du code de saisi

if code_saisi[i] <> code_acces[i] then ok:=0;

end;

if ok=1 then porte:=1 else porte:=0; // commande de la porte

end;

end.

```

## COMMANDE D'UN AFFICHEUR

### LCD

Exemple : Afficheur LCD LM016L (16x2) 2 lignes

16 colonnes

L'afficheur LM016L possède :

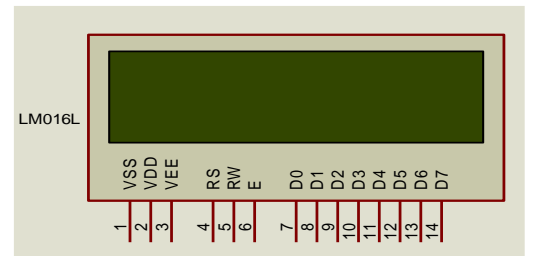
- 3 bornes d'alimentation (nommées **VSS**, **VDD** et **VEE**)
- 3 entrées de contrôle (nommées **RS**, **RW** et **E**)
- 8 entrées de données (nommées **D0**, **D1**, **D2**, **D3**, **D4**, **D5**, **D6** et **D7**)

Appelons N la valeur de l'octet de donnée D0 à D7

- **VSS** doit être reliée à la borne moins d'une alimentation de 5 V
- **VDD** doit être reliée à la borne plus d'une alimentation de 5 V
- **VEE** permet de modifier l'éclairage et le contraste de l'afficheur
- Si **RS = 0** alors N est une instruction
- Si **RS = 1** alors N est une donnée: N correspond au code ASCII du caractère à afficher
- Si **RW = 0** alors l'afficheur est en mode écriture (write)
- Si **RW = 1** alors l'afficheur est en mode lecture (read)
- **E** est l'entrée de validation activée à niveau haut

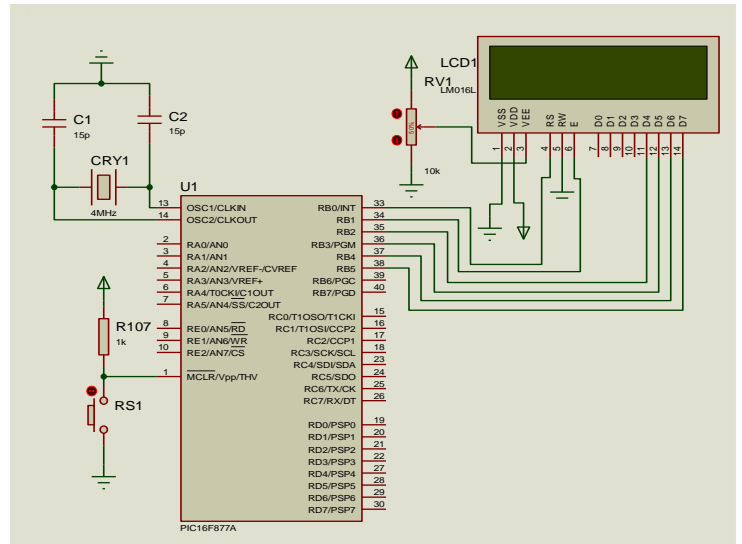
Instructions spécifique au compilateur Mikropascal pro pour l'afficheur LCD

Les variables suivantes doivent être définies dans tous les projets utilisant la bibliothèque d'affichage à cristaux liquides LCD:



## // Connections du module LCD

```
var LCD_RS : sbit at RB0_bit;  
var LCD_EN : sbit at RB1_bit;  
var LCD_D4 : sbit at RB2_bit;  
var LCD_D5 : sbit at RB3_bit;  
var LCD_D6 : sbit at RB4_bit;  
var LCD_D7 : sbit at RB5_bit;
```



```
var LCD_RS_Direction : sbit at TRISB0_bit;  
var LCD_EN_Direction : sbit at TRISB1_bit;  
var LCD_D4_Direction : sbit at TRISB2_bit;  
var LCD_D5_Direction : sbit at TRISB3_bit;  
var LCD_D6_Direction : sbit at TRISB4_bit;  
var LCD_D7_Direction : sbit at TRISB5_bit;
```

// FIN

```
Lcd_Init () ; // Initialisation de l'LCD
```

```
Lcd_Out(1, 2, 'BRAVO') ; // écrire BRAVO sur l'LCD à partir de la ligne 1 ,colonne 2
```

```
Lcd_Chr(2, 3, i); // écrire la caractère équivalent en code ASCII i sur l'LCD à partir de  
la ligne 2 ,colonne 3
```

```
Lcd_Cmd exemples :
```



```
Lcd_Cmd(_LCD_CLEAR); // effacer l’LCD
```

```
Lcd_Cmd(_LCD_CURSOR_OFF) ; // supprimer le curseur de L’LCD
```

## Activité N° 1 (LCD)

Sur un afficheur LCD « LM016L » écrire : • 4 Sc. TECHNIQUE 4 sur la 1<sup>ère</sup> ligne et la 1<sup>ère</sup> colonne

• 2014-2015 sur la 2<sup>ème</sup> ligne et la

4<sup>ème</sup> colonne

• # au 2<sup>ème</sup> ligne et 14<sup>ème</sup> colonne

Compléter puis saisir le programme ci-contre puis simuler et vérifier le fonctionnement :

```
program activite_LCD_1;
```

```
// connections du module Lcd
```

```
var LCD_RS : sbit at PORTB.0;
```

```
var LCD_EN : sbit at PORTB.1;
```

```
var LCD_D4 : sbit at PORTB.2 ;
```

```
var LCD_D5 : sbit at PORTB.3;
```

```
var LCD_D6 : sbit at PORTB.4;
```

```
var LCD_D7 : sbit PORTB.5;
```

```
var LCD_RS_Direction : sbit at TRISB.0;
```

```
var LCD_EN_Direction : sbit at TRISB.1 ;
```

```
var LCD_D4_Direction : sbit at TRISB.2 ;
```

```
var LCD_D5_Direction : sbit at TRISB.3 ;
```

```

var LCD_D6_Direction : sbit at TRISB.4;

var LCD_D7_Direction : sbit at TRISB.5;

begin

    LCD_init(); // initialisation de LCD

    LCD_CMD(_LCD_CURSOR_OFF); // supprimer le curseur de LCD

    while true do

        begin

            LCD_out(1,1,'4 sc.TECHNIQUE 4'); // 4 Sc.TECNIQUE 4 s'écrit à la 1ère ligne et
la 1ère colonne

            LCD_out(2,4,'2014-2015'); // 2014-2015 s'écrit à la 2ème ligne et la 4ème colonne

            Lcd_chr(2,14,35); // écrire la caractère # au 2ème ligne et 14ème colonne ( 35 est le code
ASCII de #)

        end;

    end.

```

## GESTION D'UN CLAVIER

Le mikroPascal PRO pour PIC fournit une bibliothèque pour travailler avec des claviers « en bloc de touches 4x4 ». Les routines de bibliothèque peuvent également être employées avec le bloc de touches 4x1, 4x2, ou 4x3.



L'utilisation de ce clavier nécessite un port obligatoirement bidirectionnel 8 bits.

Instructions spécifique au compilateur « Mikropascal pro » pour le Clavier

```
Var keypadPort : byte at PORT... ; // Pour la connexion du clavier au PORT. ..  
considéré ( 8 bits)
```

```
Var Kp :byte ; // on définit une variable de type octet
```

```
Keypad_Init(); // Initialisation du clavier
```

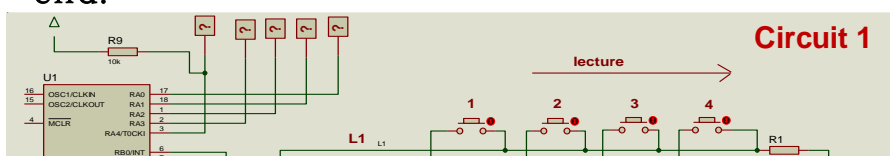
```
Kp:=Keypad_key_Press(); // lecture de code de la touche « touche enfoncée » de 1 à  
16.
```

```
Kp:=Keypad_key_click(); // lecture de code de la touche « touche enfoncée puis  
libérée » de 1 à 16.
```

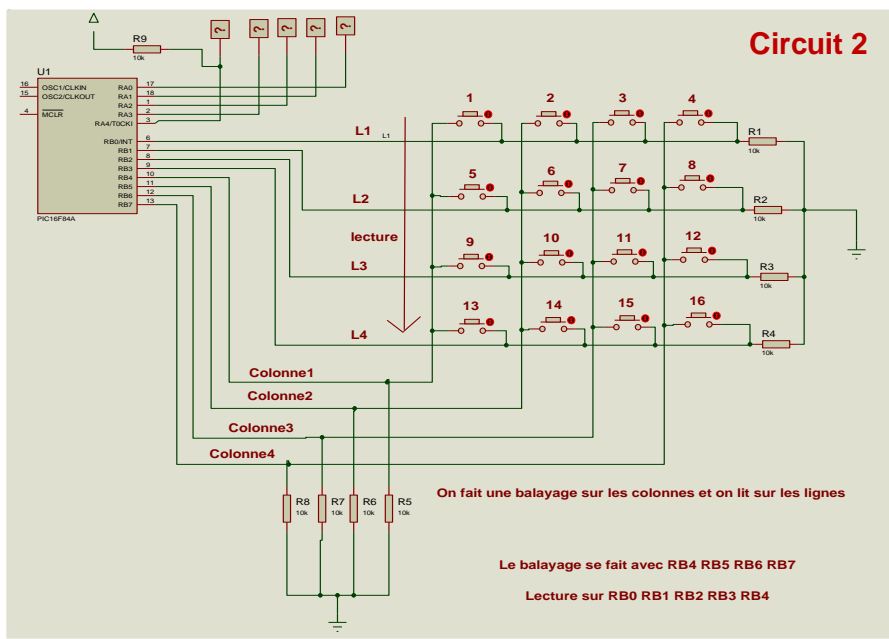
### Activité N° 1 (Clavier)

Saisir le programme suivant puis simuler le fonctionnement de chaque circuit et remplir les tableaux correspondants :

```
program clavier_prefabrique;  
var keypadPort : byte at PORTB; // le clavier est relié au port B  
var kp : byte; // On déclare une variable kp de type octet  
begin  
trisa:=0; // Le portA est configuré en sortie  
porta:=0; // Initialisation du portA  
Keypad_Init(); // initialiser le portB pour communiquer avec le clavier  
while true do  
begin  
kp := Keypad_Key_Click();  
if kp <> 0 then porta:=kp;  
end;  
end.
```



Touche	Code en décimal
1	1
2	2

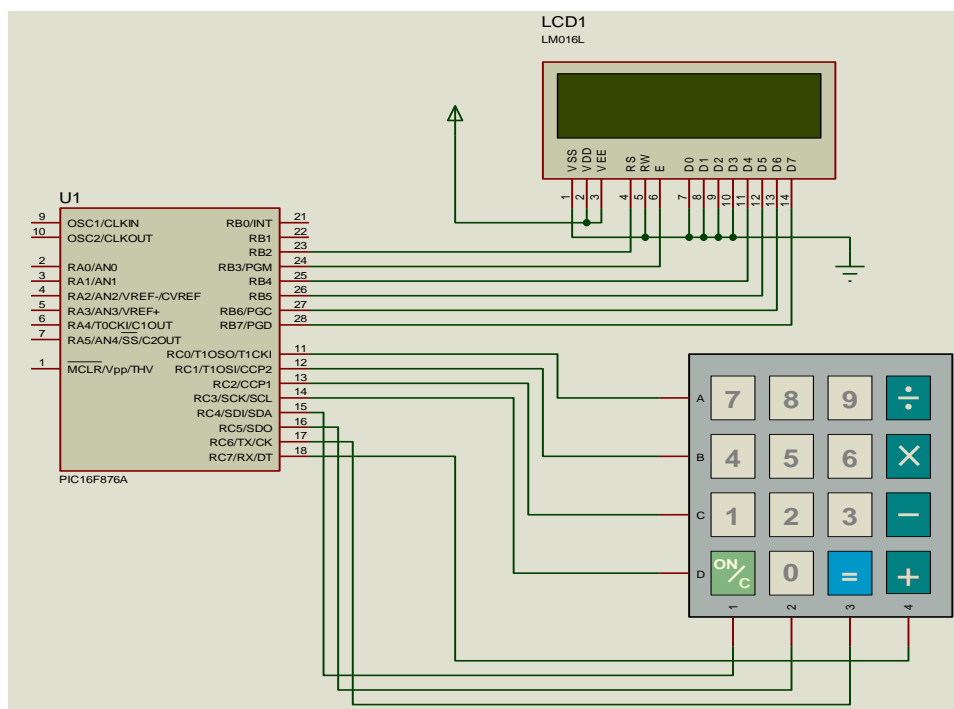


Touche	Code en décimal
1	1
2	5
3	9
4	13
5	2
6	6
7	10
8	14
9	3
10	7
11	11
12	15
13	4
14	8
15	12
16	16

## Activité N° 2 (Clavier)

Ecrire un programme qui permet d'afficher sur la première ligne et la première colonne  
La touche est :

Et en 2<sup>ème</sup> ligne et 1<sup>ère</sup> colonne Touche : le code de la touche appuyé du clavier à la  
colonne 10



```
program clavier_ex1;
```

```
var
```

```
while true do
```

```
begin
```

```
kp:=0;
```

```
while kp=0 do
```

```
begin
```

```
kp:=keypad_key_click();
```

```
kp:byte;
```

```
var keypadPORT:byte at PORTC;
```

```
var LCD_RS:sbit at RB2_bit;
```

```
var LCD_EN:sbit at RB3_bit;
```

```
var LCD_D4:sbit at RB4_bit;
```

```
var LCD_D5:sbit at RB5_bit;
```

```
var LCD_D6:sbit at RB6_bit;
```

```
var LCD_D7:sbit at RB7_bit;
```

```
var LCD_RS_direction:sbit at TRISB2_bit;
```

```
var LCD_EN_direction:sbit at TRISB3_bit;
```

```
var LCD_D4_direction:sbit at TRISB4_bit;
```

```
var LCD_D5_direction:sbit at TRISB5_bit;
```

```
var LCD_D6_direction:sbit at TRISB6_bit;
```

```
var LCD_D7_direction:sbit at TRISB7_bit;
```

```
begin
```

```
  lcd_init();
```

```
  Keypad_Init();
```

```
  lcd_cmd(_lcd_clear);
```

```
  lcd_cmd(_lcd_cursor_off);
```

```
  lcd_out(1,1,'La touche est:');
```

```
  lcd_out(2,1,'Touche:');
```

### Activité N° 3 (Clavier) « serrure codé »

Pour accéder à un immeuble, l'utilisateur doit saisir un mot de passe

à l'aide d'un clavier matriciel 12 touches installé à l'entrée.

Si le mot de passe est correct, l'ouverture de la porte est assurée par un relais qui doit rester alimenté avec le témoin vert pendant 2 secondes.

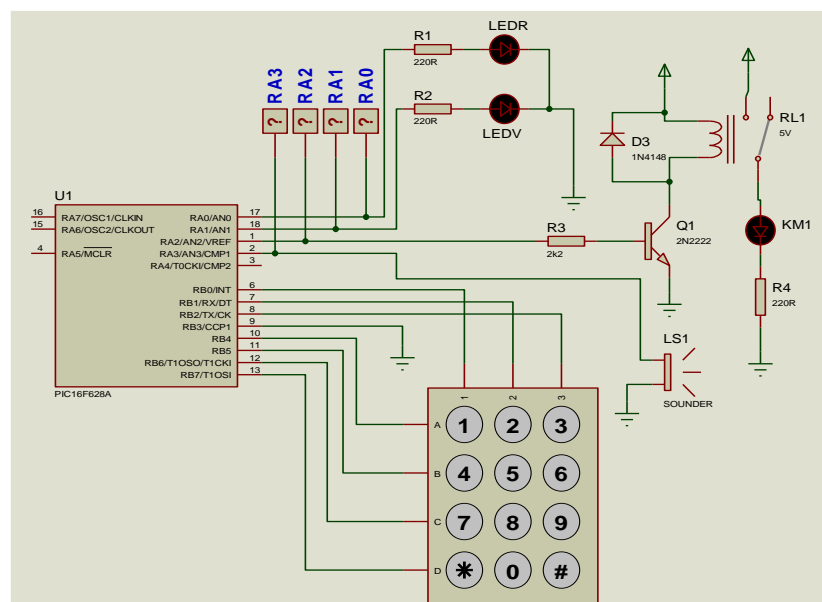
Si le mot de passe est incorrect, un témoin rouge s'allume durant une seconde, la porte reste fermée et l'utilisateur doit saisir à nouveau le mot de passe.

L'appui sur la touche \* valide le mot de passe saisi, l'appui sur la touche # permet d'initialiser le système pour une nouvelle lecture du code d'accès.

La serrure est munie d'un haut-parleur piézoélectrique «Sounder» qui émet un bip sonore à chaque appui sur une touche du clavier.



Schéma du



montage :

1° ) Saisir le programme ci-dessous et simuler le fonctionnement et déduire le rôle du programme.

```
program serrure;
var keypadPort : byte at PORTB; // le clavier est relié au port B
var kp : byte; // kp est une variable de type octet
begin
  trisa:=0; // Le portA est configuré en sortie
  porta:=0; // Initialiser le port a
  CMCON:=$07; // désactiver les comparateurs, porta numérique
  Keypad_Init();// initialiser le clavier
  while true do
    begin
      kp := Keypad_Key_Click();// Lecture de code de la touche (1,,16). Si aucune clef n'est
      cliquetée, renvoie 0
      if kp <> 0 then  porta:=kp;
      end;
    end.
end.
```



Le but de ce programme est de **coder chaque touche**

2° ) Pour chaque touche appuyée, retrouver le code correspondant en décimal.

Touche	1	2	3	4	5	6	7	8	9	*	0	#
Code en décimal	1	2	3	5	6	7	9	10	11	13	14	15

3° ) Saisir le programme ci-dessous et simuler le fonctionnement :

```
program serrure2;
```

```
var keypadPort : byte at PORTB; // Clavier est relié au PORTB
```

```
var kp,a : byte ;
```

```
code_serrure : word;
```

```
begin
```

```
trisa:=0 ; // PORTA sortie
```

```
porta:=0 ; // Initialiser le PORTA
```

```
CMCON:=$07 ; // désactiver les comparateurs PORTA numérique
```

```
code_serrure :=0 ;
```

```
a:=0; // Initialiser les variables
```

```
kp:=0;
```

```
Keypad_Init(); // Initialiser le port pour travailler avec le clavier
```

```
sound_init(porta,3); // Initialiser le module « piezo-haut-parleur » qui est affecté à
```

```
la broche RA3
```

```
while true do // boucle infinie
```

```
begin
```

```
kp := Keypad_Key_Click(); // lecture du code de la touché
```

```
if kp <> 0 then // si une touche est enfoncée
```

```
begin
```

```
sound_play(1000,100); // produire un son de fréquence 1000hz qui dure 100ms
```

case kp of // selon la valeur de kp « selon la touche enfoncée » la variable a reçoit... :

1: a := 1 ;

2: a := 2 ;

3: a := 3 ;

5: a := 4 ;

6: a := 5 ;

7: a := 6 ;

9: a := 7 ;

10: a := 8 ;

11: a := 9 ;

13: a := '\*' ; // a reçoit le code ASCII du caractère \*

14: a := 0 ;

15: a := '#' ; // a reçoit le code ASCII du caractère #

end;

if a <> '\*' then code\_serrure := code\_serrure \* 10 + a ; // s'il n'y a pas d'action sur la touche \* alors :

if a = '#' then code\_serrure := 0 ; // si on actionne la touche # on initialise le code à 0

if a = '\*' then // si on actionne la touche \* on valide le code

begin

if code\_serrure = 1978 then // si le code est correcte ici code=1978

alors

begin

porta.1:=1; // on allume le témoin vert

porta.2:=1; // on ouvre la porte

delay\_ms(2000); // temporisation de 2 secondes

end

else // si non

begin

porta.0:=1; // on allume le témoin rouge

```

delay_ms(1000); // temporisation de 1seconde
end;
code_serrure :=0; // initialiser le code à 0
end;
end;
porta:=0; // Initialiser le porta
end;
end.

```

4° ) déduire à chaque fois la valeur de la variable code\_serrure lorsque l'utilisateur appuie sur les touches données dans le tableau suivant :

Touche appuyé	Valeur en décimal de la variable code_serrure= code_serrure *10 + a	Porte ouverte	Témoin rouge	Témoin verte
#	0	non	non	non
4	$0 + 4 = 4$	non	non	non
7	$4*10 + 7 = 47$	non	non	non
8	$47*10+8 = 478$	non	non	non
1	$478*10 +1= 4781$	non	non	non
*	Validation $4781 \neq 1978$ après 1seconde 0	Non	Oui (1s)	non
1	1	non	non	non
9	19	non	non	non
7	197	non	non	non
8	1978	non	non	non
*	Validation $1978 = 1978$ après 2seconde 0	oui (2s)	non	oui (2s)
#	0	non	non	non
5	5	non	non	non
4	54	non	non	non
*	Validation $54 \neq 1978$ après 1seconde 0	non	Oui (1s)	non

5° ) Quel est le code qui permet d'ouvrir la porte, justifier votre réponse en soulignant dans le programme précédent la ligne d'instruction qui teste ce code

1978

6° ) Compiler le programme précédent et simuler le fonctionnement de la serrure codée

## FONCTION CONVERSION :

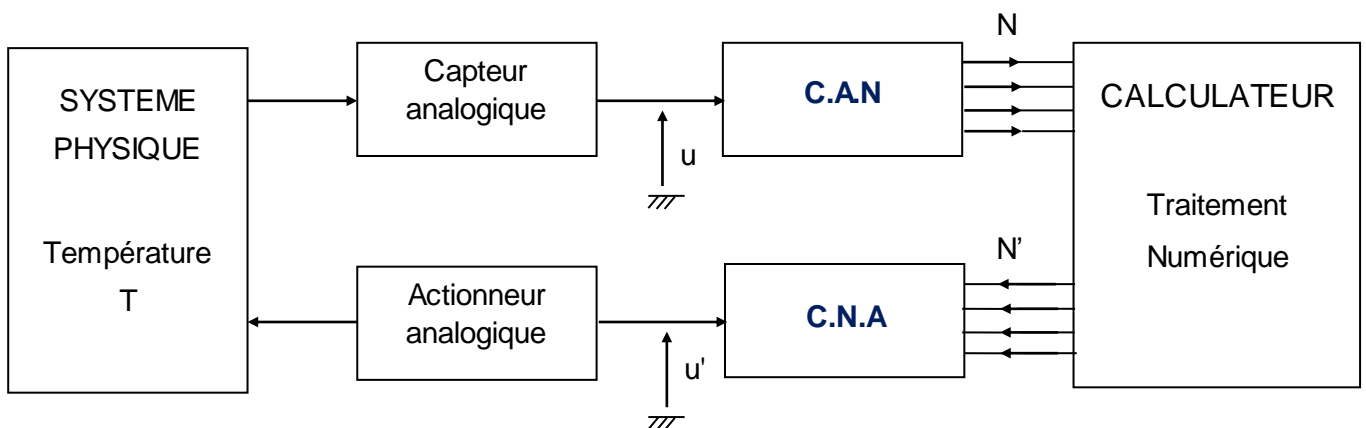
L'électronique peut se partager en deux grands chapitres : L'analogique et le numérique.

En numérique, on raisonne sur des bits portant l'information sous forme d'états logique 1 ou 0.

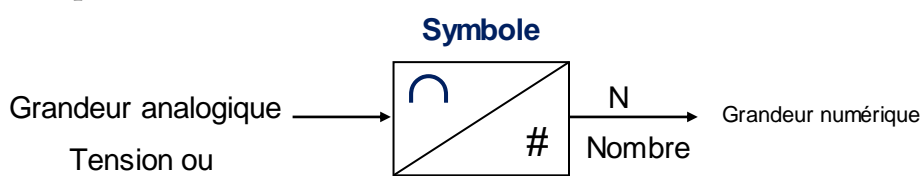
En analogique, on raisonne sur des courants ou des tensions.

Néanmoins, on est souvent amené à convertir une donnée analogique en donnée numérique afin de leur apporter un traitement numérique ou inversement.

Exemple : Régulation de température par ordinateur.



Dans cette partie on s'intéresse à étudier le C.A.N avec certains microcontrôleurs



Les microcontrôleurs PIC 16F876 et 16F877 possèdent un convertisseur analogique numérique sur 10 bits, ce dernier permet de convertir une tension analogique comprise entre  $V_{ref-}$  et  $V_{ref+}$  en une valeur numérique comprise entre 0 et 1023.

Pour exploiter ce convertisseur il est nécessaire de configurer certains registres dans le microcontrôleur,

Dans notre cas on intéresse au registre [ADCON1](#)

Registre ADCON1 :

ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
bit 7	bit 6			bit 3	bit 2	bit 1	bit 0

Bit 6, bit 5 et bit 4 : Bits non implantés.

Bit 3, bit 2, bit 1 et bit 0 : PCFG3, PCFG2, PCFG1 et PCFG0 : bits de contrôle de la configuration des ports :

Ces bits permettent de choisir le partage entre entrées analogiques et digitales sur les ports A et E .

Ils permettent également de choisir pour Vref+ entre Vdd et RA3 et pour Vref- entre Vss et RA2 selon le tableau suivant :

							PIC 16F876					Tensions De références	
4 bits PCFG				PIC 16F877									
				PORTE			PORTA						
PCFG0	PCFG0	PCFG0	PCFG0	AN7/RE2	AN6/RE1	AN5/RE0	AN4/RA5	AN3/RA3	AN2/RA2	AN1/RA1	AN0/RA0	Vref+	Vref-
0	0	0	0	A	A	A	A	A	A	A	A	V <sub>DD</sub>	V <sub>SS</sub>
0	0	0	1	A	A	A	A	Vref+	A	A	A	RA3	V <sub>SS</sub>
0	0	1	0	D	D	D	A	A	A	A	A	V <sub>DD</sub>	V <sub>SS</sub>
0	0	1	1	D	D	D	A	Vref+	A	A	A	RA3	V <sub>SS</sub>
0	1	0	0	D	D	D	D	A	D	A	A	V <sub>DD</sub>	V <sub>SS</sub>
0	1	0	1	D	D	D	D	Vref+	D	A	A	RA3	V <sub>SS</sub>
0	1	1	X	D	D	D	D	D	D	D	D	V <sub>DD</sub>	V <sub>SS</sub>
1	0	0	0	A	A	A	A	Vref+	Vref-	A	A	RA3	RA2
1	0	0	1	D	D	A	A	A	A	A	A	V <sub>DD</sub>	V <sub>SS</sub>
1	0	1	0	D	D	A	A	Vref+	A	A	A	RA3	V <sub>SS</sub>
1	0	1	1	D	D	A	A	Vref+	Vref-	A	A	RA3	RA2
1	1	0	0	D	D	D	A	Vref+	Vref-	A	A	RA3	RA2
1	1	0	1	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2
1	1	1	0	D	D	D	D	D	D	D	A	V <sub>DD</sub>	V <sub>SS</sub>
1	1	1	1	D	D	D	D	Vref+	Vref-	D	A	RA3	RA2

A : entrée analogique D : entrée numérique

$V_{DD} = V_{CC} = 5V$  ;  $V_{SS} =$

GND = 0 V

Au reset ADCON1 = 00000000 : cela signifie que les 5 bits de port A et les 3 bits de Port E sont configurés en entrées analogiques.

Pour récupérer les 5 bits du port A et les trois bits du port E en tant que I/O numériques (digitales) il faut écrire la valeur '0 6' dans ADCON1

N.B : On s'intéressera uniquement au cas où  $V_{ref-} = V_{SS} = 0$  et  $V_{ref+} = V_{DD} = 5V$  cas des lignes coloriées en rose dans le tableau.

### Bit 7 : ADFM

Le convertisseur C.A.N fournit un nombre binaire naturel de 10 bits (B9 B8 B7 B6 B5 B4 B3 B2 B1 B0)

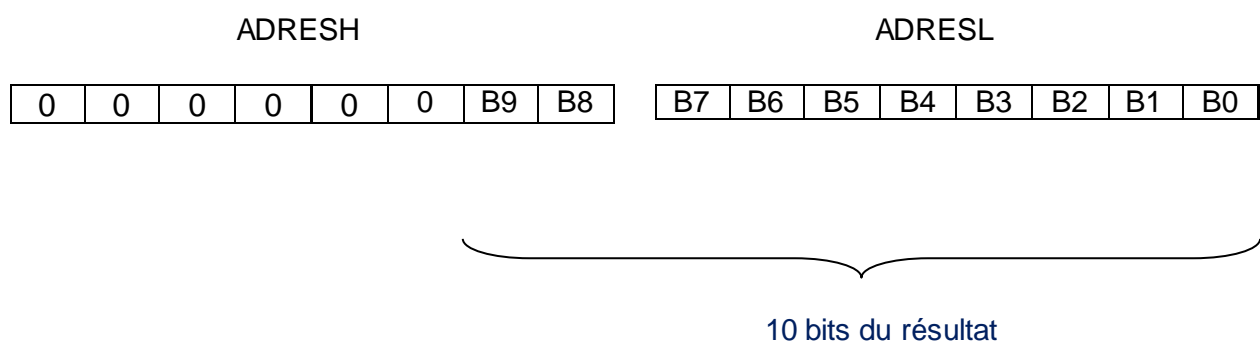
Deux registres (2 X 8 bits) sont nécessaire pour stocker le résultat de la conversion. Ce sont les registres :

- ADRESH

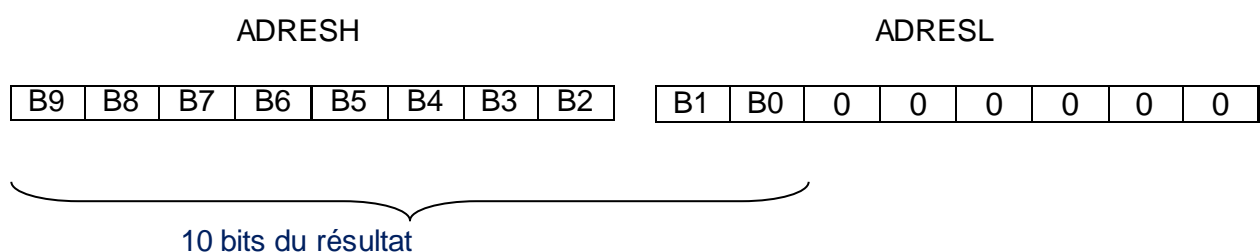
- ADRESL

Deux formats sont disponibles suivant la valeur du bit ADFM :

**ADFM=1** :le résultat de la conversion est justifié à droite :ADRESH ne contient que les 2 MSB du résultat . Les 6 MSB de ce registre sont lus comme des « 0 »



**ADFM=0** :le résultat de la conversion est justifié à gauche :ADRESL ne contient que les 2 LSB du résultat . Les 6 LSB de ce registre sont lus comme des « 0 »





Instructions spécifique au compilateur Mikropascal pro pour le module conversion  
 ADC\_Init() ; // Initialise le module convertisseur et le configurer avec les réglages  
 suivants:

Vref-=0 ;Vref+ = 5V , Utilisation de l'horloge interne pour la conversion.

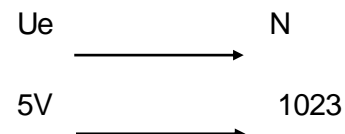
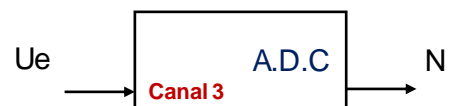
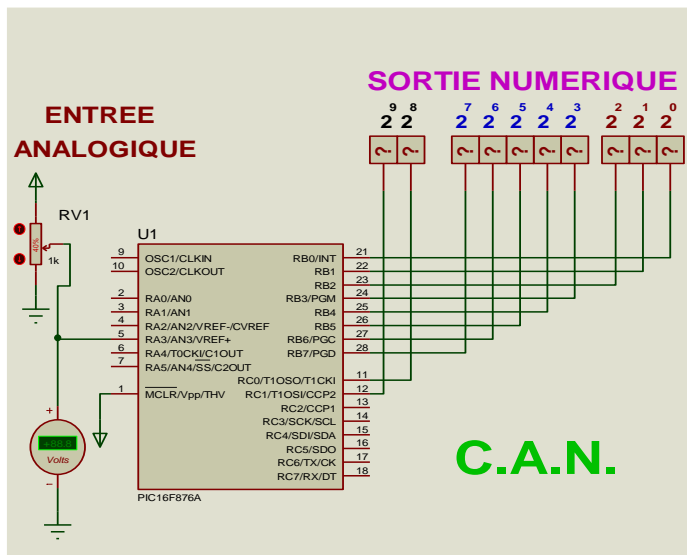
N : word // déclaration d'une variable de type word

N := ADC\_Get\_Sample(1) // lecture de la valeur lue par le convertisseur sur le  
 canal 1

N := ADC\_Read(2) // lecture après initialisation et démarrage de la conversion  
 sur le canal 2

Activité N° 1 (conversion) :

Convertir une tension comprise entre 0 et 5V fournit par un potentiomètre branché sur  
 RA3, et afficher le résultat sous forme binaire avec des LEDs. (Justification à droite)



$$N = \frac{1023}{5} U_e$$

1° ) Configurer les entrées /sorties :

TRISA						TRISB								TRISC							
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2° ) Configurer le registre ADCON 1

ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
1	0	0	0	0	0	0	0

3° ) Compléter puis saisir le programme ci-contre puis simuler et vérifie le fonctionnement :

```
program activite_1_convertisseur;
```

```
var N : word; // déclaration d'une variable N de type mot sur 16bits car le résultat de  
conversion est sur 10 bits
```

```
begin
```

```
ADCON1:=%10000000; // Configuration des entrées du portA comme entrées  
analogiques y compris RA3
```

```
// Et justification des 10 bits à droite
```

```
TRISA := $FF; // PORTA Entrées
```

```
TRISB := 0; // PORTB Sorties
```

```
TRISC := 0; // PORTC Sorties
```

```
while true do
```

```
begin
```

```
N := ADC_Read(3); // lecture de la valeur lue par le convertisseur sur le canal 3
```

```

PORTB:=ADRESL; PORTC:=ADRESH;

end;

end.

```

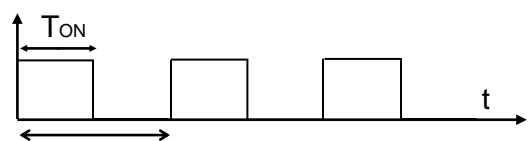
N.B. Autre solution : PORTB := N; // Les 8 bits de plus faibles poids sont aux PORTB

PORTC := N shr(8); // Afficher les 2 bits de fort poids sur RC0 et RC1

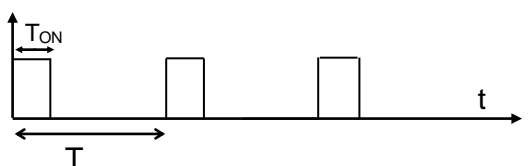
## LE MODULE MLI (PWM): PULSE WIDTH MODULATION

La modulation de largeur d'impulsion M.L.I est une technique qui consiste à générer un signal à période constante mais à rapport cyclique variable. Cette technique est largement utilisée pour faire varier la vitesse des moteurs à courant continu. La variation de vitesse d'un moteur à courant continu par M.L.I consiste à alimenter ce moteur de façon discontinue avec un hacheur et faire varier la tension moyenne entre ses bornes.

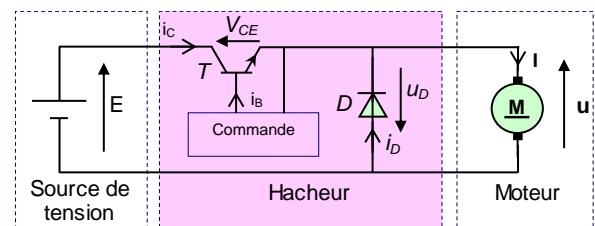
Le signal de commande du transistor est :



T Le rapport cyclique  $\alpha = \dots =$



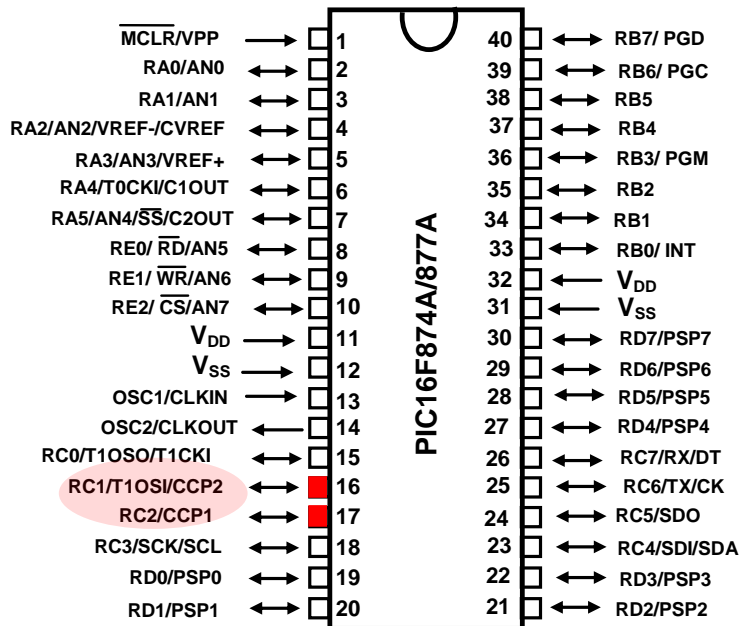
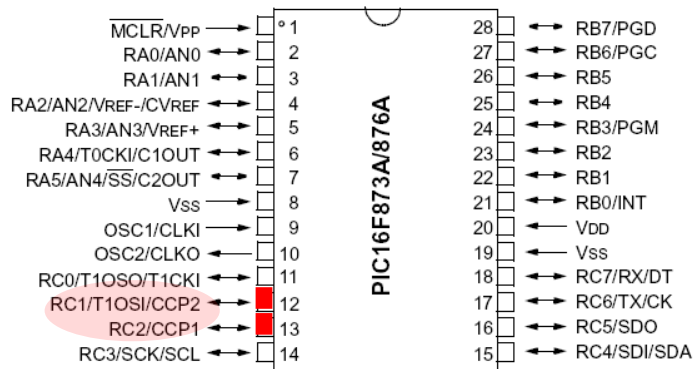
Le rapport cyclique  $\alpha = \dots =$



Plusieurs microcontrôleurs possèdent des sorties capables de générer automatiquement des signaux M.L.I généralement appelées sortie PWM notées CCP1 et CCP2 (CCP : Capture Compare Pwm)

CCP1/RC2 et CCP2/RC1  $\longrightarrow$  RC1 et RC2 sont les sorties

## 28-Pin PDIP, SOIC, SSOP



Instructions spécifique au compilateur Mikropascal pro pour le module PWM

PWMx\_Init(1000) // Initialise le module PWM de la sortie CCPx à la fréquence 1000Hz:

PWMx\_start() // Démarrage du module PWM et sortie du signal sur la broche CCPx

PWMx\_Set\_duty(N) // Change le rapport cyclique  $\alpha$  du signal sortant sur la broche CCPx avec

$$\alpha = \frac{N}{255}$$

N variant de 0 à 255

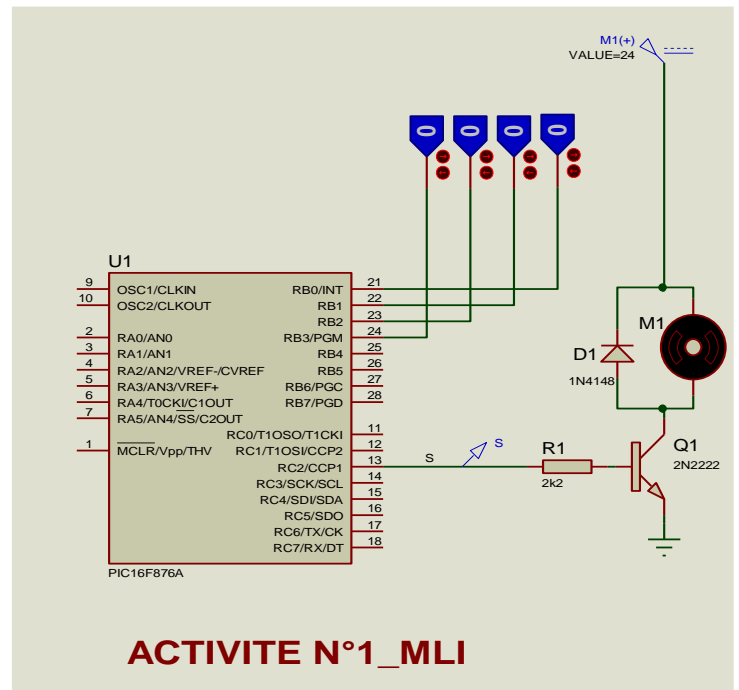
PWMx\_stop // Arrêter le module PWM de la sortie CCPx

Activité N° 1 (MLI):

Ecrire un programme qui permet de commander un moteur à courant continu avec 4 vitesses

Sortie sur RC2 fréquence de MLI (1000Hz)

Entrées	Rapport cyclique	N
PORTB =0	$\alpha = 0$	N = 0
PORTB =1	$\alpha = 0,25$	N = 64
PORTB =3	$\alpha = 0,5$	N = 127
PORTB =7	$\alpha = 0,75$	N = 192
PORTB =15	$\alpha = 1$	N = 255



```
program activite_1_MLI;
```

```
begin
```

```
    TRISB:=$FF;
```

```
    PWM1_Init(1000);
```

```
PWM1_Start;
```

```
while true do
```

```
begin
```

```
if PORTB=0 then PWM1_Set_duty(0);
```



```
if PORTB=1 then PWM1_Set_duty(64);
```



```
if PORTB= 3 then PWM1_Set_duty(127);
```

```
if PORTB= 7 then PWM1_Set_duty(192);
```

```
if PORTB= 15 then PWM1_Set_duty(255);
```

```
end;
```

```
end.
```

## Activité N° 2 (conversion et MLI): Ventilateur sur un concentrateur USB:

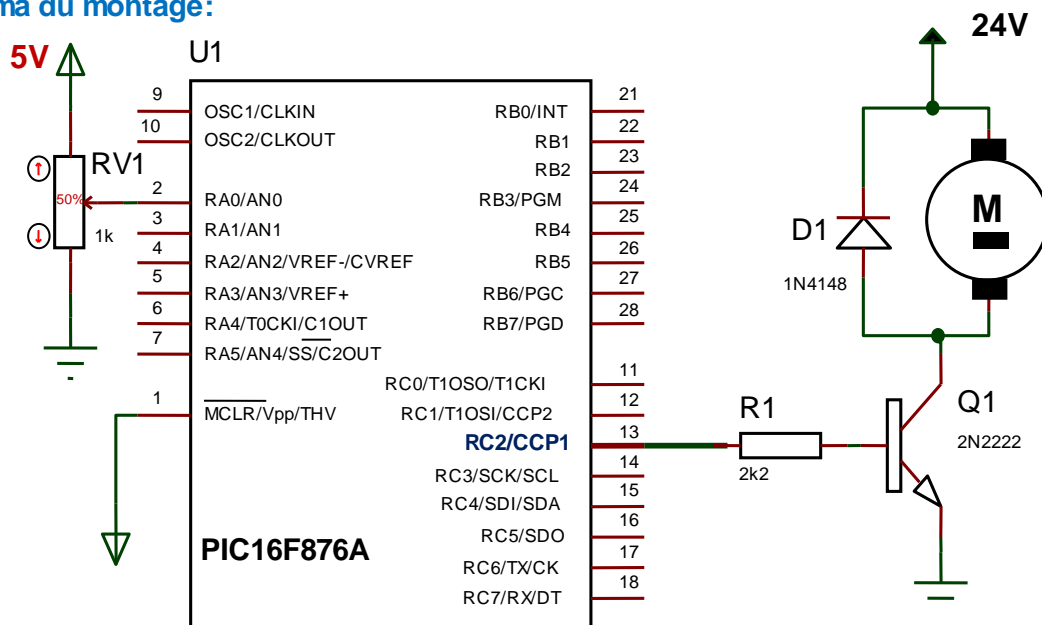
Le système présenté est un concentrateur USB appelé en anglais « USB - HUB » muni d'un petit ventilateur de bureau. Le système est autoalimenté via l'alimentation 5V de l'USB d'un micro-ordinateur.



L'hélice du ventilateur est entraînée en rotation par un moteur à courant continu de très faible puissance (5V, 200mW).

L'utilisateur a la possibilité de varier la vitesse de rotation du ventilateur via un potentiomètre rotatif « RV1 » placé à l'avant du concentrateur USB.

### Schéma du montage:



1° ) Donner le rôle du transistor Q1 :

**Le rôle du transistor Q1 est le pilotage du moteur. C'est un commutateur électronique.**

2° ) Quelle est la technique utilisée pour faire varier la vitesse du moteur M? Justifier le choix de la sortie RC2/CCP1 du microcontrôleur PIC16F876A :

C'est la technique MLI ou en anglais PWM le microcontrôleur dispose de 2 sorties PWM RC1 ou RC2

on utilise par exemple RC2.

3° ) Quel est le type du signal présent sur l'entrée RA0/AN0? Justifier le choix de cette entrée.

C'est un signal analogique variable de 0 à 5V. Il faut choisir alors une entrée analogique (RA0/AN0)

4° ) Dédurre la valeur du registre ADCON1 :

ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
1	0	0	0	0	0	0	0

5° ) N étant le nombre lu par le module convertisseur CAN du microcontrôleur et « K » étant le paramètre à passer à la fonction PWM1\_set\_duty(K), retrouver la relation entre N et K

Lorsque V varie de 0V à 5V K varie de 0 à 255 ( $\alpha$  est le rapport cyclique  $\alpha = K/255$ )  
varie de 0 à 1

N varie de 0 à 1023  $\longrightarrow$  K  
1023  $\longrightarrow$  255

Règle de trois :  $K = \frac{255}{1023} N$

6° ) Compléter le programme suivant puis simuler et vérifier le fonctionnement.

program ventilateur;

var



K : byte;

N : word ;

begin

PWM1\_init(250); // Initialiser le module PWM1 et choix de la fréquence de PWM1 =  
250 Hz

ADCON1:=\$80; // Configuration des entrées du porta comme entrées analogiques y  
compris RA0

PWM1\_start; // démarrage du module PWM1

while true do // boucle infinie

begin

N:= adc\_read(0); // lecture de la conversion

$$K := \frac{N}{4}; \quad // \text{calcul}$$

PWM1\_set\_duty( K ); // changement du rapport cyclique : le rapport cyclique  $\alpha = K/$   
255

end;

end.

## 2<sup>ème</sup> méthode

```
program activite2_MLI_conversion_bis;
```

```
var
```

```
  K : byte;
```

```
  N : word ;
```

```
  S1 :real ;
```

```
  begin
```

```
    PWM1_init(250); // Initialiser le module PWM1 et choix de la fréquence de PWM1 =  
250 Hz
```

```
    ADCON1:=$80; // $8E Configuration des entrées du porta comme entrées analogiques  
y compris RA0
```

```
    PWM1_start; // activation du module PWM1
```

```
  while true do // boucle infinie
```

```
    begin
```

```
      N:= adc_read(0); // lecture de la conversion
```

```
      S1 := 255* N/1023; // calcul
```

```
      K := byte(S1) ; // transformation du résultat de calcul en octet
```

```
      PWM1_set_duty( K ); // changement du rapport cyclique : le rapport cyclique  $\alpha = K /$   
255
```

```
    end;
```

```
  end.
```

Thanks for all