

<b>INTRODUCTION .....</b>	<b>2</b>
<b>LABORATOIRE N°1 – LES BOUTONS.....</b>	<b>3</b>
RAPPEL THEORIQUE : .....	3
EXEMPLE DE CODE : .....	3
REMARQUES IMPORTANTES : .....	3
<b>LABORATOIRE N°2 – LE CHENILLARD.....</b>	<b>4</b>
WATCHDOG – RAPPEL THEORIQUE .....	4
WATCHDOG – EXEMPLE DE CALCUL .....	4
WATCHDOG – CONFIGURATION.....	4
WATCHDOG – EXEMPLE DE CODE.....	4
WATCHDOG – REMARQUES .....	5
<b>LABORATOIRE N°3 - GESTION DU TIMER 1.....</b>	<b>6</b>
TIMER 1 – RAPPEL THEORIQUE .....	6
TIMER 1 – CALCUL DU PREDIVISEUR.....	6
TIMER 1 – PRECHARGE .....	6
TIMER 1 – LES REGISTRES.....	6
TIMER 1 – EXEMPLE DE CODE.....	7
LES INTERRUPTIONS – RAPPEL THEORIQUE .....	7
LES INTERRUPTIONS – EXEMPLE DE CODE.....	7
LES INTERRUPTIONS – REMARQUES.....	8
<b>LABORATOIRE N°4 – CONVERTISSEUR ANALOGIQUE NUMERIQUE.....</b>	<b>9</b>
CAN – RAPPEL THEORIQUE .....	9
CAN – MISE EN PLACE.....	9
CAN – EXEMPLE DE CODE.....	9
CAN – REMARQUE .....	9
<b>LABORATOIRE N°5 – COMPARATEURS.....</b>	<b>10</b>
COMPARATEURS – RAPPEL THEORIQUE.....	10
COMPARATEURS – MISE EN PLACE .....	10
COMPARATEURS – EXEMPLE DE CODE .....	10
COMPARATEURS – REMARQUE .....	10
LE MODULE VREF – RAPPEL THEORIQUE.....	10
LE MODULE VREF – MISE EN PLACE .....	10
LE MODULE VREF – EXEMPLE DE CODE .....	10
<b>LABORATOIRE N°6 – TABLES ET TIMER0 .....</b>	<b>11</b>
LE TIMER 0.....	11
LES TABLES – RAPPEL THEORIQUE .....	11
LES TABLES – EXEMPLE DE CODE .....	11
LES TABLES – REMARQUE.....	11
<b>LABORATOIRE N°7 – LIAISON RS232 .....</b>	<b>12</b>
RS232 – RAPPEL THEORIQUE.....	12
RS232 – LES REGISTRES .....	12
RS232 – ENTREES / SORTIES .....	13
RS232 – EXEMPLE DE CODE .....	13
RS232 – REMARQUES .....	13
<b>LABORATOIRE N°8 (11) – LCD .....</b>	<b>14</b>
LCD – RAPPEL THEORIQUE.....	14
LCD – MISE EN PLACE .....	14
LCD – EXEMPLE DE CODE .....	14
LCD – REMARQUES .....	14

## Introduction

Ce document à pour but de vous aider lors de la programmation de  $\mu$ controlleur 16F88.

Quelques petites remarques :

- il ne s'agit que d'un complément à la datasheet et au cours, il ne fait que reprendre les informations les plus importantes de chaque labo.
- Les informations sur certaines fonctionnalités du PIC ne sont pas complètes, il n'y a que ce qui est nécessaire pour faire les labos, sans plus.
- Les codes présentés ici ne sont que des portions de codes d'exemple, elles ne fonctionnent pas si on ne les place pas dans un code fonctionnel.
- Ce document n'a pas été corrigé par un professeur, il peut donc contenir des erreurs.

## Laboratoire N°1 – Les boutons

### **Rappel théorique :**

Sur la plaquette, il y a 3 boutons, ils sont connectés aux pattes RA0, RA1 et RA2. Le quatrième bouton fait office de reset.

Lorsque l'on appuie sur un des boutons, l'état à cette patte change jusqu'à ce qu'on le relâche. Pour savoir si un bouton est enfoncé, il faut faire un test à l'aide de **btfss**.

### **Exemple de code :**

Voici un bout de code permettant de boucler tant que le bouton est relâché et d'allumer une led quand on appuie.

Test	<b>btfss</b>	PORTA,.0	; On teste sur le premier bouton
	<b>bsf</b>	PORTB,.0	; Si il est, on allume la LED
	<b>goto</b>	Test	; Si il n'est pas enfoncé, on recommence

### **Remarques importantes :**

- Lorsqu'un bouton est enfoncé, la valeur du bit correspondant dans le registre PORTA est 0 et non pas 1. C'est pour cela qu'on utilise **btfss** et pas **btfsc**.  
Rappel : **btfss** permet de sauter la ligne si le bit testé est à 1, **btfsc**, si le bit est à 0.

## Laboratoire N°2 – Le chenillard

### **Watchdog – rappel théorique**

Le watchdog est un mécanisme servant à protéger un programme au cas où celui-ci ne s'exécuterait pas dans l'espace ou le temps imparti par le programmeur.

Dans les faits, le watchdog sert principalement à 3 choses : Vérifier que le programme est bien dans la plage d'adresses voulues, à sortir d'une boucle sans fin et à sortir du mode sleep.

Le watchdog est lié à un timer interne spécifique qui n'est pas synchronisé sur le programme ni sur un événement extérieur. Ce timer a une durée spécifique de débordement comprise entre 2 valeurs (minimum : durée certaine de débordement, et maximum : durée généralement constatée). Ce temps de débordement peut-être modifié par le prédiviseur qui est en réalité un postdiviseur pour le watchdog (bit PSA à 1 dans le registre OPTION pour associer le prédiviseur au watchdog, bits PS0 et PS2 pour modifier le prédiviseur).

### **Watchdog – exemple de calcul**

Nous voulons, par exemple, que le watchdog s'active toutes les 125ms ou presque.

On lit dans la datasheet que le temps de base est de 16,38ms (*cf. Datasheet, 15.12.1*)

Le reste du calcul pour choisir le prédiviseur est simple. On commence par l'approcher :  $125 / 16,38 = 7,6$ . On arrondit au prédiviseur supérieur, à savoir 8. Cela nous donne un temps de  $16,38 * 8 = 131\text{ms}$ .

### **Watchdog – configuration**

Il y a plusieurs étapes à faire pour mettre en place le Watchdog.

- Il faut l'activer dans les mots de configuration : on remplace **\_WDT\_OFF** par **\_WDT\_ON** pour l'activer.
- Il faut signaler que le prédiviseur sera configuré par les bits **PSA2:0** du registre **OPTION\_REG** en plaçant le bit **OPTION\_REG,PSA** à 1.
- On place la valeur du prédiviseur dans les bits **OPTION\_REG,PSA2:0**. (*cf. Datasheet, 2.2.2.2*)

### **Watchdog – exemple de code**

```
__CONFIG _CONFIG1, _LVP_OFF & _WDT_ON & _INTRC_IO  
__CONFIG _CONFIG2, _IESO_OFF & _FCMEN_OFF
```

```
ORG 0x000 ; vecteur reset  
bsf STATUS,RP0 ;Obligatoire avec 16F88 si vous voulez  
clrf ANSEL  
  
bsf OPTION_REG,PSA ; On signale que les bits PSA2:0 est le prédiviseur.  
bcf OPTION_REG,.1 ; Configuration du prédiviseur  
bcf OPTION_REG,.0 ; Idem
```

*Le watchdog est donc configuré et va se mettre en route tous les x temps.*

### **Watchdog – remarques**

- On utilise des valeurs de prédiviseur assez basse, on peut donc se permettre de les mettre dans les bits **PSA2:0**, mais si ils nous fallait des valeurs plus élevées que celles disponibles avec ce registre, on pourrait employer le registre **WDTCON** (*cf. Datasheet, 15.12.2*)
- Dans notre cas, on va mettre un sleep dans une boucle qui fera déplacer la led pour que le pic se mette en veille quand il arrive à cet endroit.

boucle1	<b>sleep</b>		
	<b>rlf</b>	PORTB,0	;déplacement à gauche
	<b>movwf</b>	PORTB	;de la led allumée
	<b>btfsc</b>	PORTB,.7	;si la led 7 est éteinte on passe dans la boucle 1
	<b>goto</b>	beep	;si pas, on va dans beep
	<b>goto</b>	boucle1	

Cela permet de faire une pause qui sera arrêtée lorsque le watchdog s'activera. Il faut cependant faire attention à ce que le temps d'exécution des instructions entre deux sleep ne dépasse pas le temps du watchdog (en tenant compte du prédiviseur). Dans notre cas la boucle est tellement petite que cela ne change rien.

## Laboratoire N°3 - Gestion du timer 1

### Timer 1 – rappel théorique

Le Timer 1 est un module qui compte des événements tels que des cycles d'horloge, des impulsions sur une patte du PIC, des instructions, ... A chaque fois il incrémente un registre sur 16 bits.

Lorsqu'il déborde, il génère une interruption.

Il y a plusieurs moyens d'utiliser le Timer 1 :

- Scrutation : on boucle en vérifiant en boucle si le timer déborde (passage à 1 d'un bit).
- Interruption : en autorisant les interruptions sur le timer, chaque fois qu'il débordera, le PIC se placera à l'adresse 0x04h pour y exécuter le code.

### Timer 1 – Calcul du prédiviseur

Pour comprendre plus facilement je vais utiliser un exemple : on veut qu'il déborde toutes les 500ms. De base il déborde après  $65535 * 1 / (\text{fréquence horloge interne} / 4)$ .

Avec une fréquence interne de 4Mhz, cela veut dire que le timer déborde toutes les 65 535µs.

On voudrais qu'il déborde au minimum toute les 500 000µs, le prédiviseur est donc de :  $500\ 000 / 65\ 535 = 7,62$ . En arrondissant au nombre supérieur, cela fait 8.

### Timer 1 – précharge

Si on utilise uniquement le prédiviseur par 8, on remarque que le timer déborde toutes les  $65\ 535\mu s * 8 = 524\ 280\mu s$ , soit de trop. Pour éviter ce problème, on va entrer le surplus dans les registres de comptage du timer, de sorte qu'il ne lui reste plus qu'à compter pendant 500 000µs.

La valeur à entrer dans ces registres est de :  $24\ 280 / 8 = 3035$  (101111011011 en binaire).

A chaque débordement du timer il faudra donc mettre 11011011 dans **TMR1L** et 00001011 dans **TMR1H**.

### Timer 1 – les registres

Il y a 3 registres pour le timer1 : les deux registres de comptage et le registre de configuration.

**TMR1H** et **TMR1L** : ces deux registres sont les registres de comptage. Il y en a deux car le timer fonctionne sur 16 bits et que les registres du 16F88 font 8 bits. **TMR1H** est la partie haute et **TMR1L** est la partie basse.

**T1CON** : registre de configuration du timer.

**T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)**

U-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

Les bits qui nous intéressent sont T1CKPS1 et T1CKPS2, on y entre les valeurs du prédiviseur. (cfr. *Datasheet*, 7.1) et **TMR1ON** qui doit être placé à 1 pour activer le timer.

## Timer 1 – exemple de code

```

bcf      TRISA,.4      ;buzzer en sortie
bsf      PIE1,TMR1IE  ;autorisation des interruptions
bsf      OSCCON,.6    ;pour le timer1
bsf      OSCCON,.5    ;Interruptions
banksel  PORTA        ;On se place dans la bank0
movlw    b'11000000'  ;Autorisation des interruptions de
movwf    INTCON       ;GIE ET PEIE

movlw    b'00110001'  ;Configuration du timer1 On et
movwf    T1CON;       ;clockselect

clrf     PORTB        ;On vide le portB
bsf      PORTA,.4    ;On allume le buzzer
boucle goto  boucle    ;On tourne en rond en attendant le timer.

```

*On entre dans une boucle infinie après configuration du timer et des interruptions sur celui-ci. Lorsque le timer débordera, le PIC exécutera le code inscrit à l'adresse 0x04 et reviendra dans la boucle après en attendant un nouveau débordement.*

## Les interruptions – rappel théorique

Le principe des interruptions est très simple, lorsqu'un événement se produit, on peut choisir de déplacer l'exécution du programme à l'adresse 0x04. Pour revenir de l'interruption, il suffit de faire un **retfie**.

De base, aucun évènement n'est géré, pour les activé il faut se reporter au tableau (*cfr. Datasheet, 15.10*).

Par exemple si on veut autoriser les interruptions sur le Timer 1, il faut partir de la droite du schéma et placer les bits que l'on rencontre à 1. Cela donne : **GIE**, **PEIE** et **TMR1IE**.

Lors de l'interruption, le bit **TMR1IF** passe à 1 et génère une interruption.

*NB : Cette méthode est identique pour tout les périphériques : il faut mettre tout les bits que l'on rencontre à 1, et lors de l'interruption le bit `nom_du_periph + F` passe à 1.*

## Les interruptions – exemple de code

```

ORG 0x004          ; vecteur d'interruption
nop                ; ici prennent place les instructions de l'interruption
bcf  REGISTRE,BIT  ;ne pas oublier de remettre à 0 le bit « F » du periph
retfie              ;Fin de l'interruption

```

### ***Les interruptions – remarques***

- Lors d'une interruption, le bit GIE passe à 0 de telle sorte qu'aucune autre interruption ne puisse venir couper la routine. Lorsque l'on quitte l'interruption à l'aide de **retfie**, on revient à l'emplacement précédent l'interruption et on replace GIE à 1.
- Les bits « F » des périphériques passent à 1 lors d'une interruption, il ne faut pas oublier de les remettre à 0 lorsque le périphérique a été servi sinon lors du retour de l'interruption, il en refait une directement.

## Laboratoire N°4 – Convertisseur Analogique Numérique

### CAN – rappel théorique

Je ne vous ferait pas l'affront de vous rappeler en quoi consiste une conversion analogique/numérique, mais je vous explique comment cela se passe dans le PIC. Pour faire la conversion, il faut commencer par indiquer au convertisseur sur quelle patte arrive le signal d'entrée (c'est à dire les bits 3 à 5 du registre ADCON0). Il faut ensuite regarder si la conversion a été effectuée ou pas en faisant un test sur le bit 2 du registre ADCON0. Si ce bit est à 1, la conversion est en cours et s'il est à 0 la conversion est finie. Quand la conversion est faite, le résultat se retrouve dans les registres ADRESH et ADRESL (le résultat est codé sur 10 bits au total).

### CAN – mise en place

La mise en place du convertisseur est assez simple, il faut juste faire bien attention à effectuer les diverses opérations dans un ordre bien précis (*cf. Datasheet, 12.0*)

- On place dans **ANSEL** les bits correspondant aux pattes qui serviront pour la conversion à 1. Ex : si on veut une conversion sur la patte RA0 du PIC, on place 00000001 dans **ANSEL**.
- On configure le registre **ADCON1** (*cf. Datasheet, 12.3*). Le bit **ADFM** permet de spécifier si on veut une justification à gauche ou à droite dans les registre **ADRESH** et **ADRESL**. Cela vient du fait que la conversion est effectuée sur 10 bits et que l'on en a 16 de disponibles. Les bits **VCFG1:0** permettent de spécifier la tranche pour la conversion. (*cf. Datasheet, 12.3*)
- On configure le registre **ADCON0** (*cf. Datasheet, 12.2*) Je vous renvoie à la datasheet pour les détails concernant les différentes configurations possibles.
- On place le bit **ADCON0,2** à 1 pour démarrer la conversion.

A ce stade le convertisseur effectue une conversion, lorsqu'elle est finie le bit **ADCON0,2** passe à 0 et on peut afficher le résultat.

### CAN – exemple de code

	<b>bsf</b>	STATUS,RP0	;Sélection de la bank 1
	<b>movlw</b>	b'00000001'	;Valeur à mettre dans le ANSEL
	<b>movwf</b>	ANSEL	;pour sélectionner l'entrée digitale (p113)
	<b>clrf</b>	ADCON1	;Justifier à gauche, division de l'horloge désactivée
			;Référence de tension : Vref+ = Vdd, Vref- = Vss
	<b>clrf</b>	TRISB	;leds en sortie
	<b>bcf</b>	STATUS,RP0	;Sélection de la bank0
	<b>movlw</b>	b'11000001'	;valeurs pour ADCON : horloge dérivée de l'interne
	<b>movwf</b>	ADCON0	;Conversion sur RA0, module en fonctionnement (p114)
dem	<b>bsf</b>	ADCON0,2	;On démarre la conversion
boucle	<b>btfss</b>	ADCON0,2	;Test si une conversion est en cours (p114)
	<b>goto</b>	affichage	;si non, on affiche
	<b>goto</b>	boucle	;si oui, on boucle

### CAN – remarque

On peut essayer de lire la valeur dans les registres **ADRESH** et **ADRESL** avant que la conversion ne soit terminée mais le résultat ne sera pas correct.

## Laboratoire N°5 – comparateurs

### Comparateurs – rappel théorique

Dans le 16F88 il y a deux comparateurs qui peuvent être branchés de manières différentes. Il y a 8 branchements différents. (cfr. Datasheet, 13.1)

Dans tout les cas, les sorties des comparateurs sont les bits **C1OUT** et **C2OUT** du registre **CMCON**. La configuration est déterminée par les bits **CM2:0** du registre **CMCON**.

On peut inverser les sorties à l'aide des bits **C1INV** et **C2INV** du registre **CMCON**.

### Comparateurs – mise en place

La mise en place des comparateurs est assez simple puisqu'il suffit de configurer le registre **CMCON**. (cfr. Datasheet, 13.0)

### Comparateurs – exemple de code

<code>movlw</code>	<code>b'00000110'</code>	<code>;On configure les comparateurs</code>
<code>movwf</code>	<code>CMCON</code>	<code>;pour travailler entre RA0 et VRef</code>

### Comparateurs – remarque

Les comparateurs ont un certains temps de réponse dont il faut tenir compte (cfr. Datasheet, 18.0). Dans notre cas, je met cela en remarque car le temps est nettement inférieur à la période de l'horloge interne que nous utilisons, du coup le temps entre deux instructions est plus grand que le temps de réaction des comparateurs.

### Le module Vref – rappel théorique

Il s'agit d'un module permettant de générer une tension comprise entre Vss et Vdd, en fonction de sa configuration.

Les sorties peuvent être soit interne (sur les comparateurs) ou externe (sur la patte RA2).

### Le module Vref – mise en place

Tout comme pour les comparateurs, la mise en place est assez simple. Il y a juste quelques petites opérations à effectuer :

- Choisir si on veut connecter la sortie à RA2 avec le bit **CVRCON,CVROE**.
- Choisir si on veut travailler entre 0.25 et 0.75 \* Vsrc ou 0.00 et 0.75 \* Vsrc avec le bit **CVRCON,CVRR**.
- Mettre en route le module en place le bit **CVRCON,CVREN** à 1.

A ce stade on peut sélectionner la tension de sortie à l'aide des bits **CVRCON,CVR3:0**. Si on augmente la valeur inscrite dans ces bits, on augmente la tension de sortie et inversement.

### Le module Vref – exemple de code

<code>movlw</code>	<code>b'11000000'</code>	<code>;On met VRef au minimum</code>
<code>movwf</code>	<code>CVRCON</code>	<code>;-&gt;entre 1/4 et 3/4 Val</code>
<code>bsf</code>	<code>CVRCON,.3</code>	<code>;on place 16 dans CVR, la tension de</code>
		<code>; sortie vaut la moitié de la tension max</code>

## Laboratoire N°6 – tables et timer0

### Le Timer 0

Le timer 0 fonctionne de la même manière que le timer 1 a quelques différences près :

- il fonctionne sur 8 bits plutôt que 16
- le prédiviseur se configure dans **OPTION\_REG**

Remarque : attention à ne pas oublier de mettre le bit **OPTION\_REG,PSA** à 0 pour affecter le prédiviseur au timer 0 et non pas au watchdog.

### Les tables – rappel théorique

La table créée dans le programme est une succession d'instructions qui vont, à chaque fois que l'on rentrera dans la table, nous renvoyer une valeur.

Pour ce faire, il faut qu'à chaque fois que l'on rentre dans la table celle-ci nous renvoie la valeur suivante.

2 instructions vont nous le permettre :

- **Addwf** : Ajoute la valeur contenue dans **W** au registre **F**.

Syntaxe : **addwf f** où **f** est le registre dont la valeur contiendra (**w+f**)

- **Retlw** : Instruction qui permet de sortir d'une sous routine avec une valeur spécifiée dans le registre **W**.

Syntaxe : **retlw k** où **k** est une valeur.

Pour savoir quelle ligne de la table doit être renvoyée, nous utilisons donc **addwf** sur la variable **PCL** qui équivaut au numéro de ligne qui est en cours d'exécution par le µcontrôleur. Il n'y a plus qu'à mettre dans **W** une valeur qui rajoutée à **PCL** nous enverras sur la ligne souhaitée.

**Retlw** renverra la valeur voulue.

### Les tables – exemple de code

```
table  addwf PCL
      retlw .5
      retlw .12
      retlw .21
      retlw .37
      retlw .42

      movlw .2      ;on met 2 dans W
      call  table  ;Après cette ligne on a la valeur 12 dans W
```

### Les tables – remarque

Les tables peuvent être utilisées dès que l'on a un ensemble de valeur, ou pourrait les comparer aux tableaux dans les autres langages de programmation.

En C on mettrait :

*W = table[2]*

En assembleur :

**Movlw** .2

**Call** table

## Laboratoire N°7 – Liaison RS232

### **RS232 – rappel théorique**

L'USART (Universal Synchronous Asynchronous Receiver Transmitter) permet d'envoyer et de recevoir des informations en série et ce de manière synchrone ou asynchrone.

Nous travaillerons en asynchrone. Pour envoyer ou recevoir une donnée, tout se fait sur des registres, il suffit d'écrire ou de lire les registres.

### **RS232 – les registres**

Il y a deux registres pour configurer l'USART, et un troisième qui permet de régler la vitesse de transmission.

#### **Registre TXSTA (registre de transmission)**

**CSRC** : permet de déterminer quelle horloge sera utilisée. Inutile en mode asynchrone.

**TX9** : permet de choisir le mode de transmission : 8 bits ou 9 bits (0 = 8bits). Nous choisissons 8 bits.

**TXEN** : permet de signifier que l'émission est activée. On positionne ce bit à 1.

**SYNC** : permet de choisir entre synchrone ou asynchrone. On choisit asynchrone en le positionnant à 0.

**BRGH** : permet de choisir si on va travailler en high speed ou low speed pour le baud rate generator. Dans notre cas on le positionne à 0 pour low speed.

**TRMT** : bit permettant de savoir l'état de la transmission. Quand il est à 0 cela veut dire qu'une transmission est en cours.

**TX9D** : 9ème bit si ce mode de transmission est choisi. Inutile dans les autres cas.

Pour ce laboratoire le registre **TXSTA** contiendra : 00100000

#### **Registre RCSTA (registre de réception)**

**SPEN** : Activation du port série. Cela permet de configurer les pins 2 et 5 du portb pour la transmission/réception de données séries. Il sera mis à 1.

**RX9** : Permet de définir si la réception se fera sur 8 ou 9 bits, nous choisissons 8 bits (bit à 0)

**SREN** : Réception simple, inutile en mode asynchrone, ce bit restera à 0.

**CREN** : bit d'activation de la réception continue. Ce bit sera mis à 1 pour activer la réception de données.

**ADDEN** : Bit de détection d'adresses : permet d'activer le module de détection d'adresses, il sera mis à 0 pour le désactiver.

**FERR** : bit d'erreur.

**OERR** : bit d'erreur.

**RX9D** : 9ème bit dans le cas où la réception est configurée sur 9 bits, si pas il est inutile.

Pour le labo la configuration sera : 10010000

Registre **SPBRG** : c'est le registre du baud rate generator, le module permettant de signaler la vitesse de transmission des données.

En fonction de la fréquence de l'horloge interne qui a été configurée et de la vitesse voulue on place une certaine valeur dans ce registre. Les formules pour savoir quelle est la valeur à mettre sont :

Il y a aussi des tableaux de valeurs pré calculées pour nous aider (*cf. Datasheet, 11.3*)

## **RS232 – entrées / sorties**

### **La réception :**

Le mode de réception, une fois l'USART activé, est très simple : lorsqu'une donnée est reçue, une interruption est générée et la valeur de la donnée reçue se trouve dans le registre **RCREG**, il suffit alors de la copier de ce registre au registre où elle est nécessaire.

### **La transmission :**

Pour transmettre une valeur avec l'USART, il suffit de placer cette valeur dans le registre **TXREG**, la transmission s'effectuera automatiquement

## **RS232 – exemple de code**

```
clrf ANSEL ; Utilisation PORTA en E/S digitales
bsf INTCON,GIE ; Autorisation des interruptions générales
bsf INTCON,PEIE ; Autorisation des interruptions périphériques
bsf PIE1,RCIE ; Autorisation des interruptions sur la réception de l'USART
movlw b'00000100' ; Port b en sortie
movwf TRISB ; Sauf pour les pattes 2 et 5
movlw .51 ; On place 25 dans le baud rate generator
movwf SPBRG ; pour avoir une vitesse de 2400bauds
bsf TXSTA,TXEN ; On active la transmission de l'USART
bcf STATUS,RP0 ; Banque 0
bsf RCSTA,SPEN ; On active le transfert série
bsf RCSTA,CREN ; On active la reception continue.

movlw 'A' ; On choisit la valeur à envoyer
movwf TXREG ; On place la valeur à envoyer dans le registre d'emission de l'USART

movf RCREG,1 ; On place la valeur de la reception dans W
```

## **RS232 – remarques**

- Les bits 2 et 5 de **TRISB** doivent être positionnés à 1 pour que les transmissions se fassent.
- Lorsque l'on envoie une donnée, il faut bien attendre que l'octet ait été envoyé avant d'inscrire le suivant dans le registre **TXREG**, pour cela il suffit de scruter le bit **PIR1,TXIF**.
- Lors de la réception d'une donnée, le bit **RCIF** passe à 1, on peut donc vérifier la réception soit par scrutation, soit par interruption.

## Laboratoire N°8 (11) – LCD

### *LCD – rappel théorique*

On utilise des fonctions pré programmées pour manipuler ce LCD. Elles sont disponibles dans les fichiers LCD8BIT et LCD4BIT en fonction du mode de fonctionnement voulu.

Il y a trois fonctions que nous utiliserons :

- **Sendchar** : Cette routine envoie le caractère contenu dans **W** au LCD
- **Sendcmd** : Cette routine envoie la commande contenue dans **W** au LCD
- **LCDInit** : Routine d'initialisation du LCD

Les autres fonctions ont déjà été vues auparavant.

### *LCD – mise en place*

Pour initialiser et utiliser un LCD, il faut passer par une série d'étapes :

- appeler la fonction **LCDInit** pour initialiser le LCD dans le mode choisi.
- utiliser les fonctions **Sendcmd** ou **Sendchar** pour envoyer des informations.

### *LCD – exemple de code*

```
call    LCDInit      ;Initialisation du LCD et affichage des caractères fixes
movlw   .86          ; V
movwf   LCD_CHAR    ; on place la valeur dans LCD_CHAR pour l'envoyer
call    SendChar     ; Utilisation de sendchar pour envoyer le caractère contenu dans
LCD_CHAR
movlw   b'00010101' ;Une case vers la droite
movwf   LCD_CHAR    ; on place la valeur dans LCD_CHAR pour l'envoyer
call    SendCmd      ; On utilise SendCmd pour envoyer le contenu de LCD_CHAR au LCD
en lui spécifiant qu'il s'agit d'une commande et non d'un caractère à afficher.
movlw   .61          ; =
movwf   LCD_CHAR
call    SendChar
```

A ce stade on a « V = » affiché à l'écran.

### *LCD – remarques*

- pour pouvoir utiliser ces fonctions il faut soit inclure le fichier les contenant soit le copier/coller.
- ne pas oublier de modifier les lignes de configuration du fichier inclus pour que les valeurs **LCD\_DATA**, **LCD\_CMD**, **E**, **RW** et **RS** soient adaptées au matériel employé.