La programmation des PIC en C

Les fonctions, les interruptions.

Réalisation : HOLLARD Hervé.
http://electronique-facile.com
Date : 26 août 2004
Révision : 1.2

Sommaire

Introduction

Les microcontrôleurs PIC de la société Microchip sont depuis quelques années dans le "hit parade" des meilleures ventes. Ceci est en partie dû à leur prix très bas, leur simplicité de programmation, les outils de développement que l'on trouve sur le NET.

Aujourd'hui, développer une application avec un PIC n'a rien d'extraordinaire, et tous les outils nécessaires sont disponibles gratuitement. Voici l'ensemble des matériels qui me semblent les mieux adaptés.

Ensemble de développement (éditeur, compilateur, simulateur) :
MPLAB de MICROCHIP **http://www.microchip.com**
Logiciel de programmation des composants:
IC-PROG de Bonny Gijzen **http://www.ic-prog.com**
Programmateur de composants:
PROPIC2 d'Octavio Noguera voir notre site **http://electronique-facile.com**

Pour la programmation en assembleur, beaucoup de routines sont déjà écrites, des didacticiels très complets et gratuits sont disponibles comme par exemple les cours de **BIGONOFF** dont le site est à l'adresse suivante **http://abcelectronique.com/bigonoff**.

Les fonctions que nous demandons de réaliser à nos PIC sont de plus en plus complexes, les programmes pour les réaliser demandent de plus en plus de mémoire. L'utilisateur est ainsi à la recherche de langages "évolués" pouvant simplifier la tâche de programmation.

Depuis l'ouverture du site **http://electronique-facile.com**, le nombre de questions sur la programmation des PIC en C est en constante augmentation. Il est vrai que rien n'est aujourd'hui disponible en français.

Mon expérience dans le domaine de la programmation en C due en partie à mon métier d'enseignant, et à ma passion pour les PIC, m'a naturellement amené à l'écriture de ce **didacticiel**. Celui-ci se veut **accessible à tous** ceux qui possèdent une petite expérience informatique et électronique (utilisation de Windows, connaissances minimales sur les notions suivantes : la tension, le courant, les résistances, les LEDs, les quartz, l'écriture sous forme binaire et hexadécimale.).

Ce cinquième fascicule vous permettra de programmer de façon plus structurée grâce aux fonctions. Vous pourrez enfin utiliser au mieux le PIC 16F84 grâce aux interruptions.

Structure de ce document

Ce document est composé de chapitres. Chaque chapitre dépend des précédents. Si vous n'avez pas de notion de programmation, vous devez réaliser chaque page pour progresser rapidement.

Le type gras sert à faire ressortir les termes importants.

Vous trouverez la définition de chaque **terme nouveau** en **bas de la page** où apparaît pour la première fois ce terme. *Le terme est alors en **italique**.*

La couleur **bleue** est utilisée pour vous indiquer que ce **texte** est **à taper** exactement sous cette forme.

La couleur **rouge** indique des **commandes informatiques** à utiliser.

Le matériel nécessaire

Les deux logiciels utilisés lors du premier fascicule.

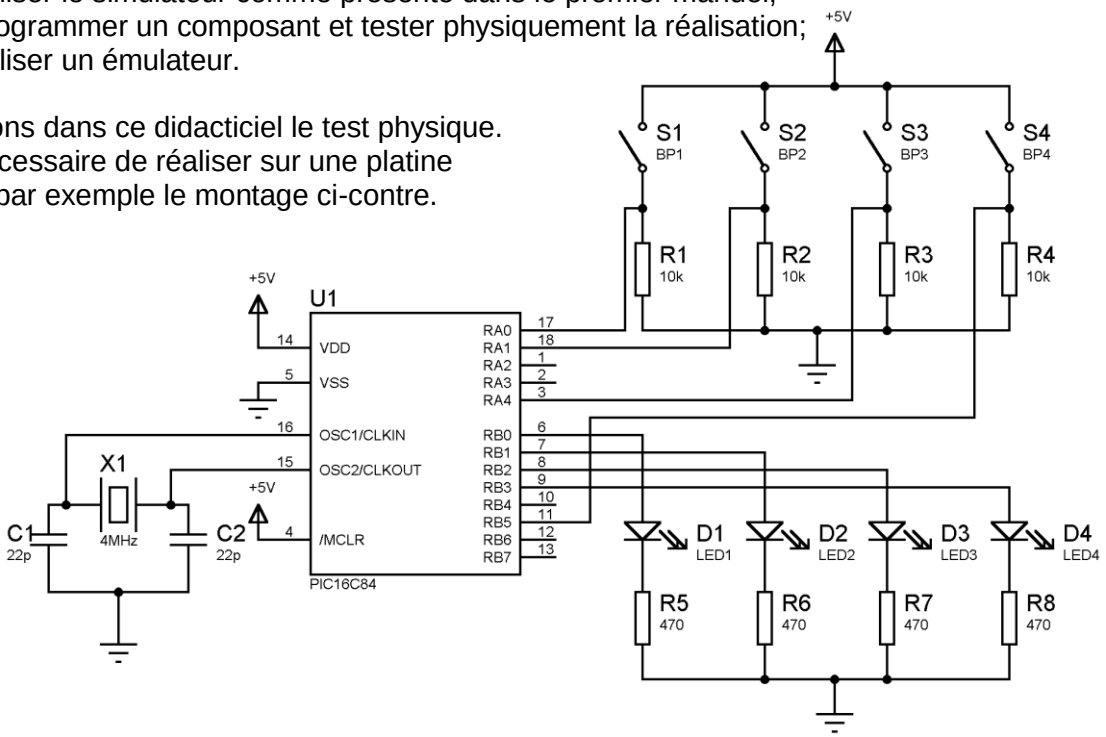
Un programmateur de PIC comprenant un logiciel et une carte de programmation. Vous trouverez tout ceci sur notre site.

Un PIC 16F84, un quartz de 4MHz, deux condensateurs de 22pF, 4 leds rouges, 4 résistances de 470 ohms, 4 interrupteurs, 4 résistances de 10 Kohms. Une platine d'essai sous 5 volts.

La platine d'essai

Pour tester les programmes proposés, il est possible :
-utiliser le simulateur comme présenté dans le premier manuel;
-programmer un composant et tester physiquement la réalisation;
-utiliser un émulateur.

Nous utiliserons dans ce didacticiel le test physique.
Il est ainsi nécessaire de réaliser sur une platine
de type LAB par exemple le montage ci-contre.



Les fonctions

Généralités

La plupart des langages de programmation nous permettent de subdiviser nos programmes en sous-programmes plus simples et plus compacts : les fonctions. A l'aide de ces structures nous pouvons **modulariser** nos programmes pour obtenir des solutions plus élégantes et plus efficaces.

Une fonction peut en appeler une autre grâce à des instructions particulières. A la suite d'un appel, **la fonction appelante s'arrête** et **la fonction appelée se met en exécution** jusqu'à ce qu'elle appelle une autre fonction ou se termine. Lors de la fin de son exécution, la fonction appelante reprend son fonctionnement depuis son arrêt.
Une fonction peut envoyer les données à la fonction qu'elle appelle au moment de son appel.
Une fonction peut envoyer des données à la fonction qui l'a appelée au moment du retour.
Une fonction appelée doit toujours être définie avant la fonction qui l'appelle. Elle est soit écrite, soit déclarée avant la fonction qui l'appelle.
Il est possible d'imbriquer jusqu'à 8 fonctions au maximum à cause de la structure du PIC 16F84. Chaque fonction peut être appelée plusieurs fois si nécessaire.

Lors du lancement du programme, la fonction "main" s'exécute en premier. C'est pour cela que nous avons écrit avant la première instruction qu'exécutera le PIC : "void main(void) {". Nous avons ainsi déclaré la fonction main. Vous comprendrez l'utilité des "void" plus tard.

Un programme en C aura donc la structure suivante:

```
Fonction X
        { Instruction en C }

Fonction Y
        { Instruction en C, appel de la fonction X si nécessaire}

Déclaration de la fonction Z

Fonction main
        { Instruction en C, appel des fonctions X, Y et Z si nécessaire}

Fonction Z
        { Instruction en C, appel des fonctions X et Y si nécessaire}
```

Les fonctions permettent :

- Meilleure lisibilité
- Diminution du risque d'erreurs
- Possibilité de tests sélectifs.
- Dissimulation des méthodes
  Lors de l'utilisation d'une fonction il faut seulement connaître son effet, sans devoir s'occuper des détails de sa réalisation.
- Réutilisation d'une fonction déjà existante
  Il est facile d'utiliser des fonctions que l'on a écrites soi-même ou qui ont été développées par d'autres personnes.
- Simplicité de l'entretien

Pour la suite du didacticiel, nous ne parlerons que du C pour CC5X. Le fond restera valable pour tous les langages C, mais des différences peuvent exister d'un langage à l'autre (obligation de positionner les fonctions après main, types de données non compatibles …).

Fonction sans paramètres

Une fonction sans paramètres **réalise une suite d'opérations sans échanger aucune donnée** avec le reste du programme.

Structure d'une telle fonction :

```
void nom_de_la_fonction(void)
        {
                // Corps de la fonction (ensemble des instructions qui constituent la fonction)
        }
```

Déclaration d'une telle fonction :

```
void nom_de_la_fonction(void);
```

Utilisation de cette fonction :

```
instruction x;
nom_de_fonction();
instruction y;
nom_de_fonction();
```

L'instruction x se réalise, puis les instructions du corps de la fonction, ensuite l'instruction y, et pour finir une nouvelle fois les instructions du corps de la fonction. Ainsi la fonction s'est réalisée deux fois, mais nous ne l'avons écrite qu'une fois. Elle ne figurera en mémoire qu'une seule fois aussi.

Exemple de programme: un compteur d'actions.

- Notre nouveau programme comptera le nombre de fois que l'on a appuyé sur le bouton poussoir 1.
- A chaque appui une nouvelle led s'allume. Nous aurons donc au maximum 4 actions.

La difficulté vient du fait que les interrupteurs ont du rebond. C'est à dire que chaque fois que l'on appuie ou que l'on relâche un bouton poussoir, l'interrupteur rebondit et le microcontrôleur comptabilise plusieurs actions. Nous allons donc écrire une fonction "anti-rebond" qui sera tout simplement une temporisation afin d'attendre que le contact s'immobilise.

Nous réaliserons aussi une fonction affichage qui mettra à jour les leds.

Dans le programme ci-dessous, la fonction "anti-rebond" est écrite après la fonction qui l'appelle, elle doit donc être déclarée avant. Ce n'est pas le cas de la fonction "affichage".

```
// Attention de respecter les majuscules et minuscules
//---------------------------------E/S-------------------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;

//----------------------------Variables globales-------------------------------
char action;                    //nombre d'actions
unsigned tempo : 16;            //tempo pour l'anti-rebond
```

```
//-------------------------------Déclaration de fonction-------------------------
void antirebond(void);

//-------------------------------Fonction affichage------------------------------
void affichage(void)
{
        sortie=0;                       // affichage du nombre d'actions
        if (action>0) led1=1;
        if (action>1) led2=1;
        if (action>2) led3=1;
        if (action>3) led4=1;
}

//-------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation du microcontrôleur
        action=0;
        TRISB = 0b11110000;
        for (;;)   // La suite du programme s'effectue en boucle
          {
                if (inter1) { ++action; antirebond(); while (inter1); antirebond(); }
                affichage();
           }
}

//-------------------------------Fonction anti-rebond----------------------------
void antirebond(void)
{
        for (tempo=0;tempo<2000;tempo++);     // anti-rebond de 20ms
}
```

Explications de la ligne : `if (inter1) { ++action; antirebond(); while (inter1); antirebond(); }`

A chaque fois que inter1 passe à 1 on réalise :

- incrémentation de "action" car on a une fois de plus enclenché le bouton poussoir.
- réalisation de "anti-rebond" pour attendre que le contact soit immobilisé à l'état 1.
- attente que le bouton poussoir soit relâché grâce à l'instruction while(inter1);.
- réalisation de anti-rebond pour attendre que le contact soit immobilisé à l'état 0;.

Si le programme comptabilise mal, augmenter la durée de l'anti-rebond.

Fonction avec paramètres d'entrée

Dans une fonction avec paramètre d'entrée, **la fonction appelée reçoit une ou plusieurs données** appelées "paramètres". Ces paramètres seront stockés au moment de l'appel de la fonction dans des variables dites temporaires qui seront détruite à la fin de la réalisation de la fonction. La notion de variable sera complètement traitée dans le fascicule suivant.

Ce type de fonctionnement **ne permet** donc **pas de modifier les données initialement envoyées** car une variable intermédiaire est créée. Nous parlerons alors de passage par données. Pour pouvoir modifier le paramètre initial, il faudra utiliser un passage par adresse, qui nécessite un pointeur et qui sera traité dans le didacticiel sur les tables.

Structure d'une telle fonction avec deux paramètres :

```
void nom_de_la_fonction(type1 nom1; type2 nom2)
    {
        // Corps de la fonction (ensemble des instructions qui constituent
        la fonction) pouvant utiliser nom1 et nom2.
    }
```

Le début de la structure est identique à la fonction sans paramètres, seule la parenthèse est modifiée. Le void qui signifiait "pas de paramètres" est maintenant remplacé par les paramètres d' entrée et leurs types. Ici le paramètre nom1 sera de type1 et le paramètre nom2 sera de type2.

Les paramètres peuvent être de n'importe quel type (bit, char, unsigned 16, …)

Déclaration d'une telle fonction :

```
void nom_de_la_fonction(type1 nom1; type2 nom2);
```

Utilisation de cette fonction :

```
instruction x;
nom_de_fonction(p,q);
instruction y;
nom_de_fonction(r,s);
```

p, q, r, s peuvent être un nombre, une variable. Il est indispensable que p et r soient compatibles avec type1, et que q et s soient compatibles avec type2.

Lors du premier appel, la fonction crée les variables nom1 et nom2. Elle copie ensuite p dans nom1 et q dans nom2. Le corps de la fonction peut se réaliser avec les variables nom1 et nom2 qui contiennent p et q. A la fin de la réalisation de la fonction, les variables nom1 et nom2 sont détruites.

Lors du deuxième appel, la fonction crée à nouveau les variables nom1 et nom2, place r dans nom1 et s dans nom2. Le corps de la fonction peut se réaliser à nouveau avec les variables nom1 et nom2 qui contiennent cette fois r et s. A la fin de la réalisation de la fonction, les variables nom1 et nom2 sont détruites.

La fonction n'a été écrite qu'une fois en mémoire, mais a été exécutée deux fois et avec des données différentes.

Exemple de programme : un compteur de points

- Quatre équipes s'affrontent et doivent chacune effectuer 5 actions.
- Chaque fois qu'une équipe a réalisé une action, elle appuie sur son bouton poussoir pour incrémenter son compteur. La valeur du compteur est alors affichée sur les leds (1 led par action).
- Lorsqu'une équipe a réalisé la 5^{e} action, la led associée à l'équipe s'allume, et le jeu est terminé.

```
// Attention de respecter les majuscules et minuscules
//-------------------------------------E/S-------------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;

//----------------------------Variables globales-------------------------------
char equipe1;               //point de l'équipe 1
char equipe2;               //point de l'équipe 2
char equipe3;               //point de l'équipe 3
char equipe4;               //point de l'équipe 4
unsigned tempo :16;     //Tempo d'antirebond

//-------------------------------Déclaration de fonction-------------------------
void affichage(char equipe);
void antirebond(void);
void fin(char gagnant);

//-------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie =0;                          // Initialisation du microcontrôleur
        equipe1=0;
        equipe2=0;
        equipe3=0;
        equipe4=0;
        TRISB = 0b11110000;
        for (;;)   // La suite du programme s'effectue en boucle
          {
                if (inter1) { if (++equipe1==5) fin(1);
                              affichage(equipe1);                       // affichage des points de l'équipe 1
                              antirebond(); while (inter1); antirebond();}
                if (inter2) { if (++equipe2==5) fin(2);
                              affichage(equipe2);
                              antirebond(); while (inter2); antirebond();}
                if (inter3) { if (++equipe3==5) fin(3);
                              affichage(equipe3);
                              antirebond();while (inter3); antirebond();}
                if (inter4) { if (++equipe4==5) fin(4);
                              affichage(equipe4);
                              antirebond();while (inter4); antirebond();}
            }
}

//-------------------------------Fonction affichage------------------------------
void affichage(char equipe)
{
        sortie=0;                           // affichage des points
        if (equipe>0) led1=1;
        if (equipe>1) led2=1;
        if (equipe>2) led3=1;
        if (equipe>3) led4=1;
}

//-------------------------------Fonction anti-rebond----------------------------
void antirebond(void)
```

```
{
        for (tempo=0;tempo<2000;tempo++);     // anti-rebond de 20ms
}

//------------------------------Fonction fin------------------------------
void fin(char gagnant)
{
        sortie=0;                       // affichage du gagnant
        if (gagnant==1) led1=1;
        if (gagnant==2) led2=1;
        if (gagnant==3) led3=1;
        if (gagnant==4) led4=1;
        for (;;);
}
```

L'instruction if (++equipe1==5) fin(1); réalise les opérations suivantes:

- Incrémentation de equipe1.
- Comparaison de "equipe1" après incrémentation avec 5 pour réaliser fin(1) en cas d'égalité.
- Fin(1) éclaire la led1 afin de signaler que l'équipe 1 a gagné.

La fonction affichage est identique à celle du programme précédent à la différence qu'on lui fournit le nombre à afficher. Nous remarquons que "equipe1", "equipe2", "equipe3", "equipe4" de type char sont tout à fait compatibles avec équipe de type char aussi.

Il est possible d'utiliser comme paramètre de fonction un nom de variable déjà utilisé. Le compilateur utilisera alors la dernière variable temporaire créée pour la fonction.

Exemple :

Nous allons utiliser la variable "equipe1" en tant que paramètre pour la procédure affiche.

Voici les modifications a apporter.

```
//------------------------------Déclaration de fonction-------------------------
void affichage(char equipe1);


//------------------------------Fonction affichage------------------------------
void affichage(char equipe1)
{
        sortie=0;                       // affichage des points
        if (equipe1>0) led1=1;
        if (equipe1>1) led2=1;
        if (equipe1>2) led3=1;
        if (equipe1>3) led4=1;
}
```

Il pourrait y avoir ambiguïté avec la variable globale appelée aussi "equipe1". Pourtant le programme fonctionne de façon identique au programme initial. Il a donc bien utilisé la variable temporaire dans la fonction et non pas la variable globale.

Ceci permet par exemple de créer plusieurs fonctions, qui réalisent des calculs mathématiques différents tout en utilisant des noms de paramètres identiques tels que "multiplicateur", "multiplicande", "resultat" …

Ayez, s'il vous plaît, une pensée émue pour l'auteur. Les trois dernières parties ont pris une trentaine d'heures pour être rédigées. Si vous trouvez qu'elles ne sont pas assez claires, prenez des pincettes pour m'en informer, je peux devenir violent.

Fonction avec paramètres de sortie

Dans une fonction avec paramètres de sortie, **la fonction appelée retourne une donnée** au programme. Cette donnée est appelée paramètre de sortie.

Structure d'une telle fonction sans paramètres d'entrée :

```
type nom_de_la_fonction(void)
        {
                // Corps de la fonction
        return donnée ;
        }
```

La structure est identique aux structures précédentes. Le void précédant le nom de la fonction est maintenant remplacé par le type du paramètre de sortie.

Le paramètre peut être de n'importe quel type (bit, char, unsigned 16, …), la donnée à renvoyer doit bien évidemment être compatible avec ce type.

Pour renvoyer le paramètre et mettre fin à la fonction, il suffit de noter "return" suivi de la donnée que l'on veut renvoyer. Cette ligne doit obligatoirement être la dernière de la fonction.

Il est aussi possible d'avoir plusieurs lignes "return donnée" dans une même fonction. La première ligne rencontrée met alors fin à cette fonction.

La donnée peut être un nombre, une variable, un calcul.

Déclaration d'une telle fonction :

```
type nom_de_la_fonction(void);
```

Utilisation de cette fonction :

```
instruction x;
a = nom_de_fonction();
instruction y;
```

Instruction x se réalise en premier, puis les instructions de la fonction. A la fin de la réalisation de la fonction, la donnée qui constitue le paramètre de sortie est déposée dans a. Nous en sommes à la troisième ligne, Instruction y se réalise.

a est une variable compatible avec le type de paramètre de sortie.

Exemple de programme : un dé électronique

- Le bouton poussoir 1 permet d'afficher de façon aléatoire une led.

Nous allons alors écrire une fonction qui fournira un nombre entre 1 et 4. Cette fonction sera appelée chaque fois que le bouton poussoir1 sera enclenché.

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variable-------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
```

```
char a;                         // variable tampon
char resultat;                  // chiffre à afficher

//--------------------------------Fonction hasard-----------------------------
char hasard(void)
{
        do {a=a+1; if (a==5) a=1;}
        while (TMR0>0);
        return (a);
}
//--------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                             // Initialisation du microcontrôleur
        TRISB = 0b11110000;
        OPTION = 0b11000111;                    // prédiviseur à 256    entrée : clock/4
        for(;;) {
                        if (inter1)
                                {
                                 resultat=hasard();
                                 sortie=0;
                                 if (resultat==1) led1=1;
                                 if (resultat==2) led2=1;
                                 if (resultat==3) led3=1;
                                 if (resultat==4) led4=1;
                                }
                        }
}
```

La fonction "hasard" ne nécessite aucun paramètre en entrée, par contre elle fournit un chiffre entre 1 et 4. Pour obtenir ce chiffre, nous allons incrémenter une variable 'a' de 1 à 4 et recommencer jusqu'à ce que le timer ait atteint la valeur 0. La fonction principale récupère cette valeur dans "résultat" afin d'afficher la bonne led.

Fonction avec paramètres d'entrée et de sortie

Vous avez bien évidemment deviné comment fonctionne une telle fonction, il suffit de combiner les paramètres d'entrée et le paramètre de sortie.

Structure d'une telle fonction avec deux paramètres d'entrée :

```
type nom_de_la_fonction(type1 nom1; type2 nom2)
        {
                // Corps de la fonction (ensemble des instructions qui constituent
                la fonction) pouvant utiliser nom1 et nom2.
        return donnée ;
        }
```

Déclaration d'une telle fonction :

```
type nom_de_la_fonction(type1 nom1; type2 nom2);
```

Utilisation de cette fonction :

```
instruction x;
a = nom_de_fonction(p,q);
instruction y;
```

Pour ce paragraphe, je ne proposerai pas d'exemple.

Les interruptions

Généralités

Ca y est, nous y sommes, nous entamons la notion qui va changer votre vie de programmateur.

Jusqu'ici, les fonctions étaient appelées par le programme lui-même. **Une interruption est une fonction qui se réalise lorsque un évènement se produit**, et non lorsque le programme le décide. Les évènements conduisant à une interruption dépendent du microcontrôleur.

Au moment de l'appel de l'interruption, le programme s'arrête. Le microcontrôleur sauvegarde l'adresse de la dernière instruction exécutée ainsi que les registres importants. L'exécution de l'interruption commence. A la fin de sa réalisation, les registres importants reprennent les états qu'ils avaient avant l'appel, le programme reprend à l'endroit où il s'était arrêté.

Le PIC 16F84

Chaque microcontrôleur fonctionne différemment lors d'une interruption. De plus, chaque langage C est différent face à ces mêmes interruptions. Nous allons donc assez rapidement analyser le comportement du PIC 16F84, et voir comment gérer cette ressource avec le C que nous utilisons depuis le début.

Le PIC 16F84 possède 4 sources d'interruption :

- Changement d'état des pattes RB4 à RB7;
- Débordement du timer (passage de 0XFF à 00);
- Front sur la patte INT (Cette patte est aussi la patte RB0). Le sens du front est déterminé par certains bits du registre option;
- Fin d'écriture dans la EEprom du PIC (partie de la mémoire qui ne s'efface pas en cas de coupure d'alimentation).

La gestion des interruptions passe par 3 opérations:

- déclaration du fichier utile à la gestion des interruptions;
- configuration des interruptions;
- écriture de l'interruption;

La déclaration du fichier utile à la gestion des interruptions est indispensable pour utiliser les instructions de sauvegarde des registres important au moment de l'interruption. Elle passe par l'écriture de la ligne ci-dessous à placer en début de fichier :

```
#include "int16CXX.h"
```

La configuration des 4 interruptions se fait à l'aide de bits des registres INTCON (INTerrupt CONfiguration) et OPTION, le plus souvent au début de la fonction main.

- RBIE (RB Interrupt Enable) : mis à 1, il autorise les interruptions dues au changement d'état des pattes RB4 à RB7.
- T0IE (Timer 0 Interrupt Enable) : mis à 1, il autorise les interruptions dues au timer.
- INTE : (INTerrupt Enable) : mis à 1, il autorise les interruptions dues à un front sur la patte RB0/INT.
- INTEDG : (INTerrupt EDGe) : un 1 valide les fronts montants comme source d'interruption sur RB0/INT, un 0 valide les fronts descendants.

- EEIE (EEprom Interrupt Enable) : un 1 valide la fin d'écriture dans la EEprom comme source d'interruption.
- GIE (General Interrupt Enable) : mis à 1, il permet aux interruptions de s'exécuter selon les bits précédents. Au début de l'interruption, ce bit est mis à 0 automatiquement. A la fin de l'interruption, ce bit est mis à 1 automatiquement.

Registre INTCON

	7	6	5	4	3	2	1	0
	GIE	**EEIE**	**T0IE**	**INTE**	**RBIE**			
Reset	0	0	0	0	0			

Registre OPTION

	7	6	5	4	3	2	1	0
		INTEDG						
Reset		1						

L'écriture de l'interruption est plus subtile qu'il n'y paraît.

Lors de la mise sous tension du microcontrôleur, le programme exécute en premier l'instruction à l'adresse 0 du programme. Le compilateur place à cette adresse l'instruction en assembleur "goto main". Ce qui signifie aller à la fonction "main".

Lorsqu'une interruption se produit, le microcontrôleur sauvegarde son environnement et saute à l'adresse 0X4 du programme. Il est donc indispensable d'écrire l'interruption à partir de cette adresse, puis d'écrire les autres fonctions. Ainsi les adresses 0X1, 0X2, 0X3 restent vides.

La structure prend la forme suivante :

```
#pragma origin 4

interrupt nom_de_l'interruption(void)
{
   int_save_registers
   Corps de l'interruption
   int_restore_registers
}
```

- #progma origin 4 indique au compilateur d'écrire à partir de l'adresse 4.
- interrupt nom_de_l'interruption(void) indique que cette fonction est une interruption et lui donne un nom.
- int_save_register est une instruction qui sauvegarde l'état des registres importants.
- int_restore_registers est une instruction qui restitue l'état des registres importants.

Le corps de l'interruption passe en premier par la détection de la source d'interruption. Pour cela, nous utiliserons certains bits du registre INTCON comme suit :

- RBIF (RB Interrupt Flag) : mis à 1 par le système, ce bit indique un changement d'état des pattes RB4 à RB7. Ce bit doit être remis à 0 manuellement avant la fin de l'interruption.
- TOIF (Timer 0 Interrupt Flag) : mis à 1 par le système, ce bit indique un débordement du timer. Ce bit doit être remis à 0 manuellement avant la fin de l'interruption.
- INTF (INTerrupt Flag) : mis à 1 par le système, ce bit indique l'arrivée d'un front sur la patte RB0/INT. Ce bit doit être remis à 0 manuellement avant la fin de l'interruption.

Registre INTCON

	7	6	5	4	3	2	1	0
						T0IF	**INTF**	**RBIF**
Reset						0	0	U

U = Unknown

Exemples

Notre premier exemple sera très original puisque nous allons faire clignoter la led1 et mettre en fonction la led 4 si le bouton poussoir 4 est enclenché.

```
// Attention de respecter les majuscules et minuscules
#include "int16CXX.h"
//--------------------------------E/S et variable------------------------------
char sortie @ PORTB;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led4 @ RB3;
char temps;                     //variable de gestion du temps égale à 15 au bout d'une seconde

//--------------------------------Interruption-----------------------------------
#pragma origin 4

  interrupt timer_et_RB5(void)
  {
        int_save_registers
        if (T0IF) { ++temps;                        // interruption timer, (65ms écoulées)
                    if (temps == 15)                // si 1 seconde écoulée
                          { led1=!led1;             // cligontement de led1
                            temps = 0; }
                    T0IF = 0;}
        if (RBIF) { led4=inter4;                    // changement de inter4
                     RBIF = 0;}
        int_restore_registers
  }

//------------------------------Fonction principale----------------------------
void main(void)
{       sortie = 0;                 // Initialisation du microcontrôleur
        TRISB = 0b11110000;
        TMR0 = 0;
        INTCON = 0b10101000;        // autorisation des interruptions par le timer et RB4 à RB7
        OPTION = 0b11000111;        // prédiviseur à 256    entrée : clock/4
        for(;;)  {
                 nop();
                 }}
```

La deuxième ligne du fichier inclue le fichier servant aux interruptions.

La première fonction est précédée d'une commande ordonnant au compilateur d'écrire à partir de l'adresse 0X4, qui est l'adresse de saut en cas d'interruption.

La première fonction est donc l'interruption.

Lors d'une interruption, il est nécessaire de

- sauvegarder les registres importants,
- tester la source d'interruption, réagir face à cette source, l'autoriser à nouveau,
- restaurer les registres importants.

Au début du programme principal, le registre INTCON est positionné tel qu'il puisse y avoir une interruption du timer ou de RB4, c'est à dire de inter4.

Nous allons maintenant programmer une serrure codée.

- Le code se compose grâce aux 3 boutons poussoirs 1, 2 et 3 afin de verrouiller la porte. Le code est 1322.
- Le bouton poussoir 4 permet d'ouvrir la porte lorsqu'elle n'est pas verrouillée. La LED 4 s'allume alors pendant que le bouton poussoir est enclenché.
- La LED1 clignote lorsque la porte n'est pas verrouillée, reste rouge lorsque la porte est verrouillée.

Nous voyons clairement que pendant que l'on utilise les boutons poussoirs 1, 2 et 3, des opérations se produisent : gestion de la porte, gestion de la LED4. Nous allons gérer ces actions par interruption. Cette gestion ressemble de plus étrangement au programme précédent.

Nous allons utiliser aussi une fonction anti-rebond pour éviter les rebonds des interrupteurs et une fonction attente qui vérifiera qu'après appui sur un bouton poussoir, celui-ci est bien relâché, et qui attendra qu'un nouveau bouton poussoir soit enclenché.

```
// Attention de respecter les majuscules et minuscules
#include "int16CXX.h"
//--------------------------------E/S-----------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led4 @ RB3;

//----------------------------Variables globales-------------------------------
bit ver;                        //etat de la porte 0=verrouillé  1=non verrouillé
unsigned tempo :16;             //tempo pour l'anti-rebond
char temps;                     //variable de gestion du temps égale à 15 au bout d'une seconde
char a;                         //variable tampon si nécessaire

//-------------------------------Interruption----------------------------------
#pragma origin 4

  interrupt timer_et_RB5(void)
  {     int_save_registers
        if (T0IF) {++temps;                     // interruption timer, 65ms écoulées
                  if (temps == 15)              // si 1 seconde écoulée
                        { if (!ver)led1=!led1;  // clignotement de led1 si non verrouillage
                          else led1=1;          //sinon led1 est allumée
                          temps = 0; }
                  T0IF = 0;}
        if (RBIF) {if (!ver) led4=inter4;               // changement de inter4, gestion de la
porte
                  else led4 = 0;
                  RBIF = 0;}
        int_restore_registers  }

//-----------------------------Fonction anti-rebond-----------------------------
void antirebond(void)
{  for (tempo=0;tempo<5000;tempo++); }          // anti-rebond de 50ms

//-----------------------------Fonction attente---------------------------------
void attente(void)
```

```
{                       // attente que les 3 inters soient relâchés puis un enclenche
        antirebond();
        a=0;
        do {a.0=inter1; a.1=inter2; a.2=inter3;} while (a!=0);          //attente que tout soit relâché
        antirebond();
        do {a.0=inter1; a.1=inter2; a.2=inter3;} while (a==0);          //attente qu'un inter soit enclenché
        antirebond();}

//------------------------------Fonction principale-----------------------------
void main(void)
{       sortie = 0;                                     // Initialisation du microcontrôleur
        ver = 0;
        TRISB = 0b11110000;
        TMR0 = 0;
        INTCON = 0b10101000;      // autorisation des interruptions par le timer et RB4 à RB7
        OPTION = 0b11000111;      // prédiviseur à 256    entrée : clock/4
for(;;) {
                                  // Recherche du code avec des if imbriqués
        if (inter1){ attente();
                if (inter3) { attente();
                        if (inter2) { attente();
                                if (inter2) {ver= !ver;}}}}
}}
```

Nous en resterons là pour les interruptions. Le prochain didacticiel vous permettra de comprendre parfaitement la notion de variables globale, locale, de constante. Vous pourrez aussi mettre en fonction vos équations les plus farfelues.