



Avant tout, bienvenue dans le domaine de l'embarqué ... !

Durant cet enseignement nous allons découvrir "les concepts de base" se cachant derrière le domaine et les applications embarquées. Nous partirons à la découverte du microcontrôleur (MCU) PIC18F4550 de chez Microchip. Notre travail se découpera en différentes parties :

- 1. Dans un premier temps, nous découvrirons l'environnement de développement MPLAB à travers quelques applications simples. Nous chercherons notamment à gérer les ports d'entrée/sortie (GPI/O). Les différents programmes seront développés en assembleur afin de nous familiariser avec l'assembleur Microchip pour sa famille PIC18.*
- 2. Dans un second temps, nous développerons un digicode capable d'interagir et de communiquer avec un PC. Ce travail commencera par le développement d'une librairie "C" pour piloter un afficheur LCD 2x16. Nous nous intéressons ensuite à la gestion d'un clavier matriciel puis d'une UART.*

Nous allons donc dans un premier temps nous intéresser à l'assembleur Microchip pour toute sa famille PIC18. Mais avant de commencer, voyons ce qui se cache derrière les ports d'entrée/sortie ...

1. Comment peut-on accéder et utiliser les ports ?

En ce qui concerne la gestion des ports, seuls trois registres de 8bits "par port" nous intéressent. Il s'agit des registres **TRISx**, **PORTx** et **LATx**, "x" correspondant au port utilisé (x= A, B, C, D ou E).

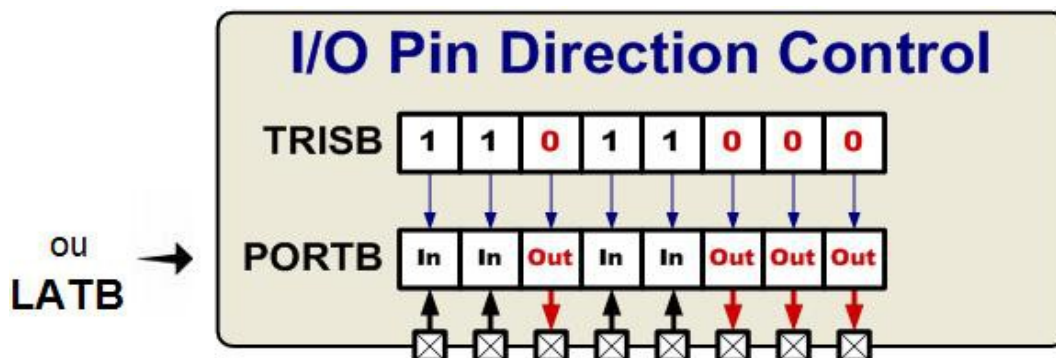


figure 1 : exemple des registres associés au port B

Avant de lire ou d'écrire sur un port, "la première chose à faire" est de fixer sa direction (en entrée ou en sortie). Prenons l'exemple du port B. Afin de mettre toutes les broches du port en sortie, il nous suffit d'écrire 0x00 dans le registre **TRISB**. Pour mettre la totalité du port B en entrée, on aurait écrit 0xFF. On peut également ne mettre qu'une partie des broches d'un port en entrée ou en sortie, comme présenté sur la **figure 1**.



*Voici un moyen mnémotechnique pour ne pas se tromper quant à la valeur à écrire dans les registres **TRIS** : "0" comme **Output** et "1" comme **Input***

Pour Lire la valeur présente sur un port, ou envoyer une donnée vers un port, nous allons utiliser les registres **PORTx** ou **LATx**. Prenons l'exemple des programmes ci-dessous qui permettent de mettre la broche n°2 du port B au niveau haut, cette broche étant configurée en sortie :

Assembleur			Langage "C"
main:	clrf	LATB	Main(){
	movlw	b'11111011'	LATB = 0x00;
	movwf	TRISB	TRISB = 0xFB;
	bsf	LATB,2	LATBbits.LATB2 = 1;

RQ : La première initialisation de LATB à "0" pourrait sembler inutile, mais cette technique est très souvent utilisée pour garantir la valeur initiale sur un port avant de fixer sa direction. Cette technique permet dans certains cas d'éviter des fronts parasites.

2. Comment accéder à une broche d'un port ?

Assembleur			Langage "C"
movlw	b'11110000'		LATB = value; // value vers LATB
movwf	LATB	;valeur 0xF0 vers LATB	PORTB = value;
movf	LATB,w	;LATB vers registre W	value = LATB ; // LATB vers value
bsf	PORTB,2	;broche 2 port B à "1"	value = PORTB ; // PORTB vers value
bsf	LATB,2	;broche 2 port B à "1"	LATBbits.LATB2 = 1; // broche 2 port B à "1"
bcf	PORTB,4	;broche 4 port B à "0"	PORTBbits.RB4 = 0; // broche 4 port B à "0"

3. Que se cache-t-il derrière une broche d'un port ?

Le schéma fonctionnel de la **figure 2** fait référence à différents éléments rencontrés durant les enseignements sur la logique combinatoire et séquentielle (bascules, portes logiques ...). La **figure 2** permet de mieux comprendre les différents mécanismes de gestion, de routage et de stockage (latch) des données au niveau d'une broche sur un port.



*Voilà ce que l'on trouve derrière **chaque** broche des ports d'entrée/sortie (GPIO) !*

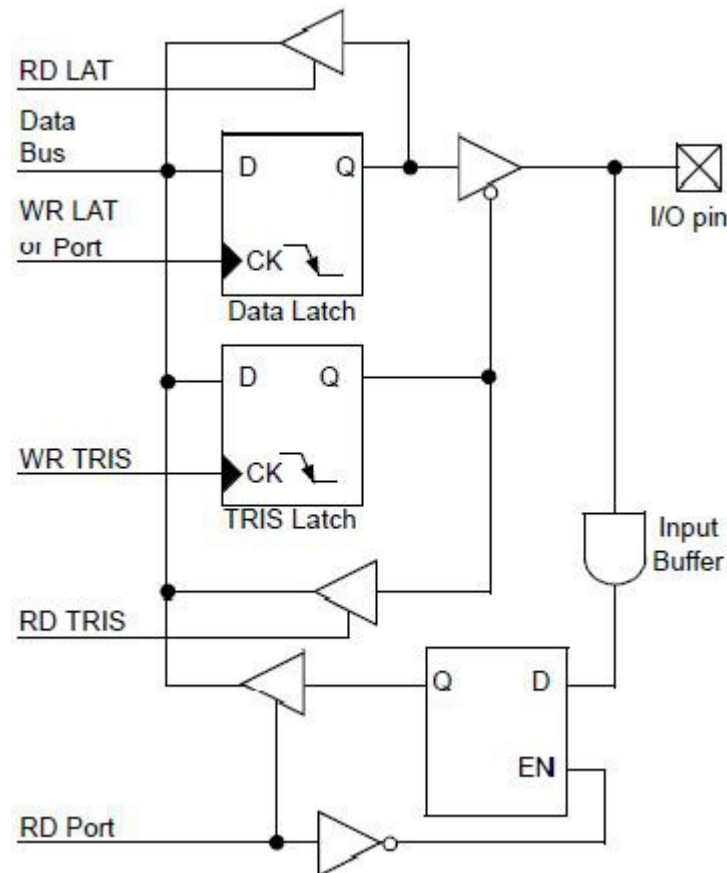


figure 2 : generic I/O port operation

4. Quand utiliser LATx plutôt que PORTx ?

Ces deux registres, LATx et PORTx sont étroitement liés et la différence entre eux deux est mince. Ils ont le même rôle, à savoir lire ou écrire une donnée à partir ou vers un port. Il faut savoir que historiquement, seuls les registres PORTx existaient. Seulement un phénomène parasite pouvait arriver lorsque l'on était amenés à faire de la manipulation de bit sur un port à la vitesse d'exécution des instructions au niveau CPU. On parle de temps cycle : Tcy. On pouvait notamment observer des valeurs erronées sur les ports.

Pour palier ce problème, les ingénieurs Microchip mirent en œuvre les registres LATx. Voici les règles à suivre pour s'assurer d'une bonne manipulation des ports d'entrée/sortie :



➔ *Si on souhaite envoyer une donnée vers un port ou des broches connectées en sorties on utilisera les registres **LATx**.*



➔ *Si on souhaite lire une donnée depuis un port ou des broches connectées en entrées on utilisera les registres **PORTx**.*



Bien, vous êtes maintenant prêts à attaquer le TP ! ... vous pouvez d'ailleurs dès à présent commencer à compléter les fichiers sources présents juste après ces questions.

Cependant, dans un soucis pédagogique, il vous est demandé un petit travail préparatoire. Il sera vérifié en début de séance et vous permettra d'anticiper une perte de temps non négligeable dans l'avancement des TP (nb ★ = difficulté).

1. (★) Quel est le rôle des registres TRISx ?

2. (★) En assembleur, proposez la suite d'instruction assurant la configuration des broches RD1 à RD4 du port D en sortie. Appliquez leurs les niveaux logiques suivants : RD3=1, RD4=1 et les autres broches à "0".

3. (★) Même question que précédemment, mais en "C" !

4. (★) Dans quel cas devons nous préférer l'utilisation des registres LATx aux registres PORTx ?



*Listing du programme **ex11.asm***

```
.*****
;
; @file : ex11.asm
; @brief :
; @author :
; last modification :
; *****

;*** Fichiers d'en-tête ***
#include <p18f4550.inc>

;*** Configuration bits ***
CONFIG WDT=OFF, LVP=OFF, PLLDIV=2, CPUDIV=OSC1_PLL2, FOSC=HSPLL_HS, BOR=OFF,...

;*** Mise en place du vecteur d'IT lié au Reset ***
org 0x0000 ; Adresse du vecteur d'IT
RESET_V:
goto MAIN ; Vecteur d'interruption lié au Reset

;*** section programme ***
code
.*****
;
;*** Programme principal ***
.*****
MAIN:

;*** Initialisation du port B ***

;*** à compléter ! ***

goto $ ; bouclage sur l'adresse courante

end
```



*Listing du programme **ex12.asm***

```
.*****
;
; @file : ex12.asm
; @brief :
; @author :
; last modification :
; *****

;*** Fichiers d'en-tête ***
#include <p18f4550.inc>

;*** Configuration bits ***
CONFIG WDT=OFF, LVP=OFF, PLLDIV=2, CPUDIV=OSC1_PLL2, FOSC=HSPLL_HS, BOR=OFF,...

;*** Section de données - initialized data in ACCESS mode ***
idata_acs
PortBTemp    db            0    ; On force le compilateur à mettre notre variable en ACCESS RAM
; Allocation d'un octet en mémoire RAM et initialisation à "0"

;*** Mise en place du vecteur d'IT lié au Reset ***
org    0x0000    ; Adresse du vecteur d'IT
RESET_V:
goto    MAIN    ; Vecteur d'interruption lié au Reset

;*** Section de code ***
code
;*****
;*** Programme principal ***
;*****
MAIN:

;*** Initialisation du port B ***

;*** à compléter ! ***

;*** Récupération valeur sur RB4-RB7 >> décalages à droite >> Affichage sur RB0-RB3 ***




;*** à compléter ! ***

end
```

Notez vos remarques :

1. Travail pratique (ex1 : partie 1) : gestion des GPI/O

Dans cette première partie de l'exercice 1, notre cahier des charges est relativement simple. Nous devons configurer les broches n°0 et n°2 du port B en sorties et leur appliquer un niveau logique haut. Voici les différentes étapes du travail à accomplir :



- ➔  Lire les documents en **ANNEXE 2** : à la découverte de MPLAB
- ➔  Lire et compléter le listing du programme **ex11.asm**.
- ➔  Créer un projet **ex1** dans votre répertoire de travail (cf. **ANNEXE 1**). Spécifier à l'assembleur d'utiliser un code **RELOGEABLE** !
- ➔ Configurer le port B
- ➔ Appliquez un niveau logique haut aux sorties correspondant au cahier des charges



A vous de jouer maintenant !

2. Travail pratique (ex1 : partie 2) : gestion des GPI/O

Dans cette seconde partie, nous allons devoir lire la valeur appliquée aux broches RB4 à RB7 (configurées en entrées) puis l'appliquer aux broches RB0 à RB3 (configurées en sorties). Par exemple, en appuyant sur les bouton poussoirs connectés aux broches RB4 et RB5, seules les LEDs connectées aux broches RB0 et RB1 s'allumeraient. Voici les différentes étapes à suivre :

- ➔  Lire et compléter le listing du programme **ex12.asm**.
- ➔  **Vous n'avez pas à créer un nouveau projet, vous devez travailler à partir du projet ex1. Retirez l'ancien fichier source du projet (Remove File From Project - ex11.asm) puis ajoutez le nouveau (Add Files to project – ex12.asm). Vous trouverez ces commandes sous l'onglet "Project".**
- ➔ Configurer le port B
- ➔ Faire tout le temps ce qui suit ...
 - ➔ Lire la valeur présente sur le port B
 - ➔ Décaler la valeur lue de 4 bits vers la droite pour préparer l'affichage
 - ➔ Afficher la valeur lue sur les broches RB0-RB3



A vous de jouer maintenant !

3. Travail pratique (*ex1 : partie 3*) : gestion des GPI/O

Réécrire l'exercice ex11.asm en C. Éditer un fichier ex11.c, le compiler puis l'exécuter. L'objectif est de prendre conscience des liens forts existants entre le C et l'asm. Les langages d'assemblages sont également des langages procéduraux. Vous aurez également l'occasion de voir quel est le fichier C minimal assurant une bonne compilation.