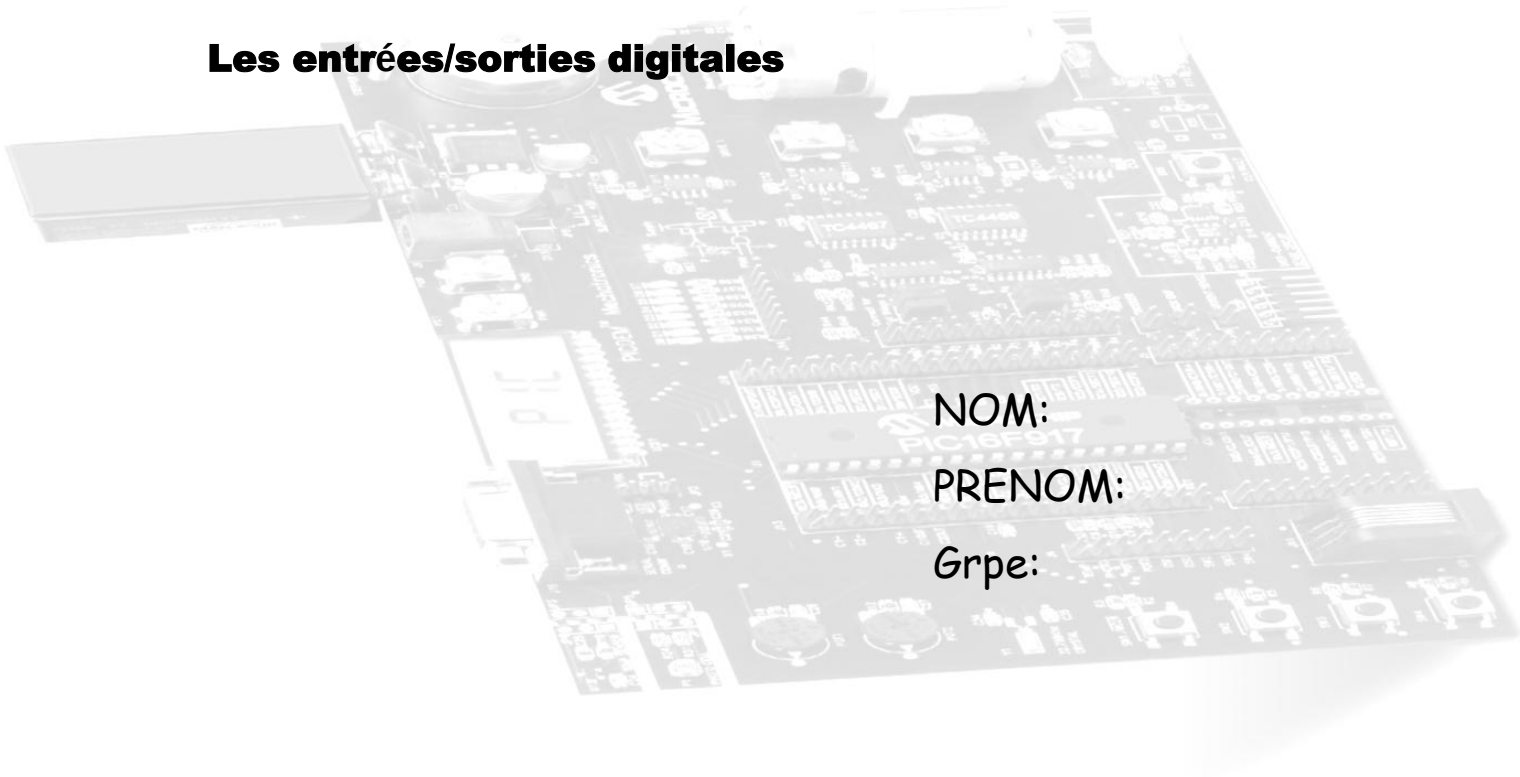
PIC

Les entrées/sorties digitales



NOM:

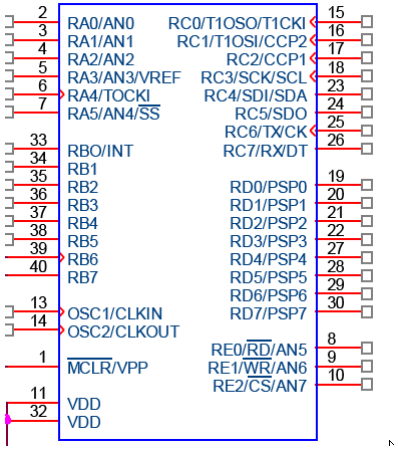
PRENOM:

Grpe:

Aspects matériels

Interfaces Parallèles

- Le micro-contrôleur PIC 18F dispose de 5 Ports d'Entrées/Sorties généraux du PORT A au PORT E. Attention les lignes des PORTs peuvent servir pour d'autres fonctions que les E/S. Le choix de la fonction se fait par programmation dans un des registres du PIC
- Ses entrées sorties permettent de commander d'autres périphériques (clavier, affichage lumineux, écran LCD, d'envoyer des ordres de commande vers d'autres composants)
- Câblage des E/S



- la configuration des broches, soit en entrées soit en sortie, est réalisé par votre programme à l'aide du registre interne: **TRISx avec x= (A,B,C,D,E)**
- **La lecture** de l'état des sorties ou des entrées est accessible à l'aide du registre interne: **PORTx avec x= (A,B,C,D,E)**
- **L'écriture** de l'état des sorties est accessible à l'aide du registre interne: **LATx**
- Au reset toutes les broches sont configurées en entrée
- le port A est configuré en mode analogique au reset du PIC (sauf RA4)

Aspects programmations

La programmation des entrées sorties

- Les différentes étapes peuvent se résumer de la manière suivante:
  - ✓ Configuration des lignes en mode digital
  - ✓ Configuration des lignes soit en entrée soit en sortie
  - ✓ Accès à l'état de la ligne
- Configuration des lignes en mode digital
  - ✓ De RB4 à RB0
    - Directives de compilation: #pragma config PBADEN = OFF // RB4:RB0 digitales au reset
  - ✓ Port C et D: digital au reset
  - ✓ Port A: analogique au reset sauf RA4. Utilisation du registre ADCON1 pour configuration digitale (voir p224 de la doc PIC18Fxx)
    - Exemples détaillés plus avan dans le cours (Thème convertisseurs)
- Configuration des lignes en entrée ou en sortie

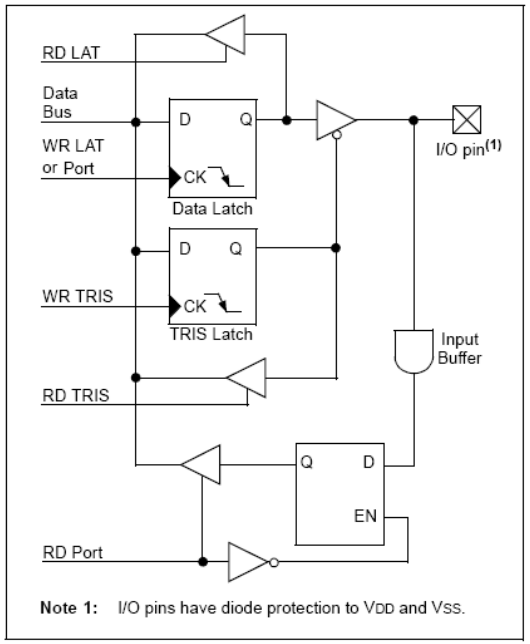
En entrée

```
TRISBbits.TRISB0 = 1;
TRISDbits.TRISD3 = 1;
```

En sortie

```
TRISAbits.TRISA1 = 0;
TRISDbits.TRISD4 = 0;
```

- Accès à l'état de la ligne
  - ✓ ***L'état d'une broche en sortie peut-être relue***
  - ✓ L'écriture sur une broche configurée en entrée est sans effet



Lecture broches configurées en entrée

Solution 1

```
var1=PORTDbits.RD3;
If(var1 == 0 ) {
..... ; // action si 0
} else {
..... ; // action si 1
```

Solution 2

```
if(PORTBbits.RB0 == 0 ) {
..... ; // action si 0
} else {
..... ; // action si 1
```

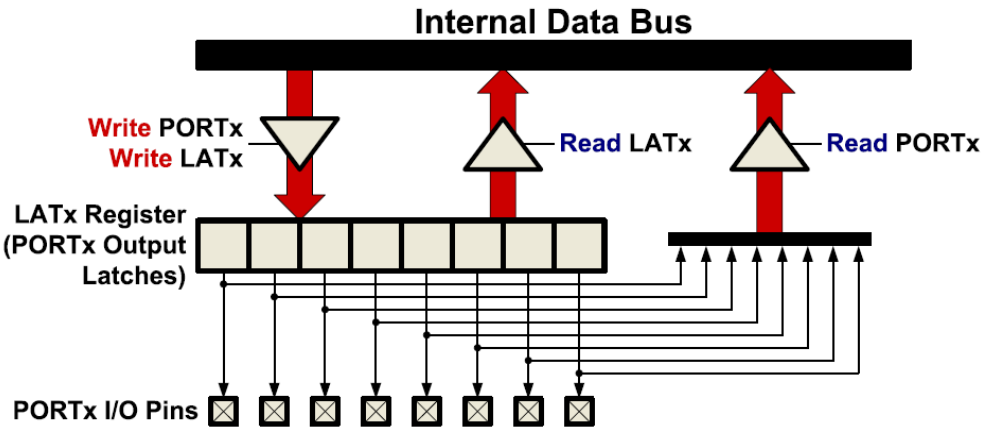
Lecture/Ecriture broches configurées en sortie

```
LATDbits.LATD4=1;
if(PORTAbits. RA1 == 0 ) {  // Relecture
..... ; // action si 1
};
```

Analogie avec *buffer* en VHDL

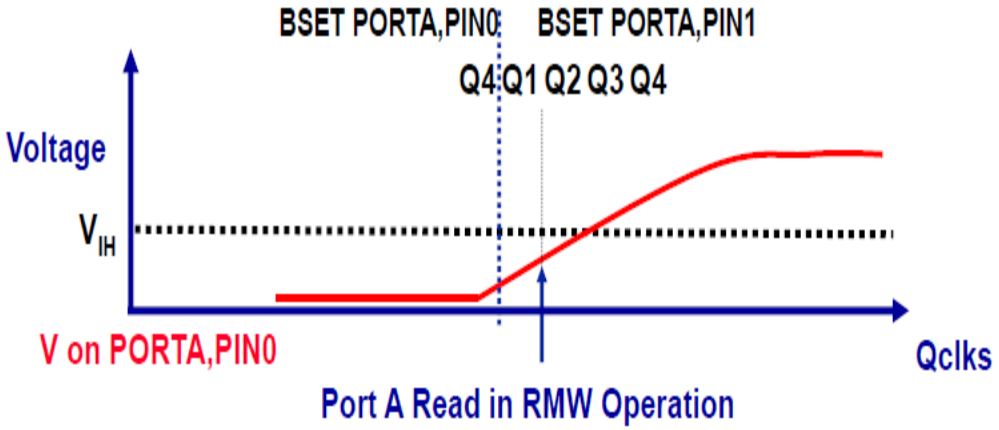
Bonne pratique: utiliser LAT

- **Différence entre LAT et PORT**
  - LAT donne accès à l'état LOGIQUE des broches (A PRIVILEGIER)
  - PORT accède à l'état PHYSIQUE des broches (permet de détecter un conflit électrique sur des broches)
  - A SAVOIR: toutes les opérations de type RMW sur 1 bit entraîne une manipulation du port en ENTIER (lecture et écriture du port dans sa totalité)



- **Problème avec l'instruction PORT**

```
Mon programme
…..
PORTAbits.RA0=1;
PORTAbits.RA1=1;
…..
```



En Q1 la TOTALITE du port A est lu. Le bit 0 est lu comme un '0' (au lieu d'un '1'). Le bit 1 est mis à 1. La totalité du port A est mis à jour
Résultat: RA0 vaut 0 et RA1 vaut 1

Programmation: exemples

Des exemples

On recopie l'état d'un bouton poussoir sur une led

```
#include <p18cxxx.h>
// configuration
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF

#define S2 PORTAbits.RA4// une manière pratique de redéfinir les bits
#define S3 PORTBbits.RB0// en fonction de leur utilisation sur la carte

#define LED2 LATBbits.LATB2
#define LED1 LATBbits.LATB1
#define LED3 LATBbits.LATB3

void main(void)
{
int cpt=0; char etat=0;
TRISBbits.TRISB1 = 0 ;  // RB1 en sortie
TRISBbits.TRISB2 = 0 ;  // RB2 en sortie
TRISBbits.TRISB3 = 0 ;  // RB2 en sortie
TRISBbits.TRISB0 = 1 ;  // RB0 en entrée S3
TRISBbits.TRISA4 =1 ;   // RA4 en entrée S2
LATBbits.LATB3 = 0 ; // mise à 0 LED3
LED2=0; // mise à 0 LED2 en utilisant le define
LED1=0;
for (;;) {
if(S2==0) // un simple interrupteur
  {
    LED2=1;
  }
  else
  {
   LED2=0;
  }
}
```

Comment compter le nombre d'appui sur une touche??

```
for (;;)  // SOLUTION 1
{
  if(S2==0)
     {compteur++;}
}
```

```
For(;;) // SOLUTION 2
{
While(S2!=0)  { //ne fait rien} ;
compteur++;
While(S2!=1)  ;
}
```

```
void main() // SOLUTION 3
{
unsigned char compteur = 0, b = 0;
for(;;)
  {
  if(S2==0)
    {
     if(b==0)
        { compteur ++;

          b=1;
        }
    }
  else
    {
     if(b==1)
         {
          b=0;
         }
    }
  } // fin for
} // fin main
```

Applications

Programmation structurée

- Permet de modéliser un scénario
  - ✓ La machine d'état

- Principe de codage

```
int etape = 0 ; //initialisation
....;
for( ; ; ) {
....;
switch (etape) {
case 0 :
// traiter étape 0 ;
break ;
case 1 :
// traiter étape 1 ;
break ;
case N :
.......;
break ;
default : // erreur !!
.......;
}
....;
}
```

- Application au comptage du nombre d'appuis

```
switch (etat)
 {
  case 0 : if(PORTBbits.RB0==0)
           {etat=1};
           break;
  case 1 : compteur++;
           etat=2;
           break;
 case 2 : if(PORTBbits.RB0==1)
          {etat=0;};
          break;
 default: etat=0;
 }
```

```
case N :
action N ;
if ( T_N_M ) {
etape = M ; }
else {
if ( T_N_P ) {
etape = P ; }
}
break ;
```

Cas d'une divergence sur plusieurs états possibles

Opérateurs logiques bit à bit

Opération ET logique bit à bit : &

- Cet opérateur sert à mettre certains bits à ZERO et à garder intacts les autres :
  - ✓ Exemple : soit trois variables de type int val1, val2, val3
    val1 = 0x34 et val2 = 0xF0 val3 = val1 & val2
    Résultat : val3 prend la valeur 0x30
- Tous les **bits dont le ET logique est fait avec un bit à zéro est mis automatiquement à zéro**
- Cette fonction est dite masquage car les bits mis à zéro sont alors non significatifs.

Opération OU logique bit à bit : |

- Cet opérateur sert à mettre certains bits à UN et à garder intacts les autres :
  - ✓ Exemple : soit trois variables de type int var1, var2, var3
  - ✓ var1 = 0xCD et var2 = 0x0F
    var3 = var1 | var2
    Résultat : var3 prend la valeur 0xCF
- Tous les **bits dont le OU logique est fait avec un bit à UN sont mis automatiquement à UN.**
- Cette fonction force un ou plusieurs bits à UN

Opération OU Exclusif logique bit à bit : ^

- Cet opérateur sert à complémenter certains bits et à garder intacts les autres :
  - ✓ Exemple : soit trois variables de type int vac1, vac2, vac3
    vac1 ← 0xAA55 et vac2 ← 0xFF00
    vac3 ← vac1 ^ vac2
    Résultat : vac3 = 0x5555
- Tous les bits dont le OU exclusif logique est fait avec un bit à un sont automatiquement complémentés, les autres bits gardant leur valeur initiale

Compléments

Travail par port

- Les exemples précédents illustrent un travail par bit
  - ✓ TRISBbits.TRISB0=0 ; LATBbits.LATB0=1; ……
- Ils est possible de travailler sur plusieurs lignes à la fois
  - ✓ Il faut être capable de configurer plusieurs lignes sans changer les autres=> UTILISER DES MASQUES
- Configuration du sens au transfert

Opération qui peut être dangereuse:il faut connaître le sens de transfert de chaque ligne

Bits	7	6	5	4	3	2	1	0
TRISB	1	0	1	0	1	1	0	1
Lignes	E	S	E	S	E	E	S	E

TRISB=valeur

Opération DECONSEILLEE

TRISB=TRISB**&**0xAD; // masque **ET** configuration de 6,4,1 en sortie (mise à 0 des bits)

TRISB=TRISB|0xAD; // masque **OU** configuration de 7,5,3,2,0 en entrée (mise à 1 des bits)

A faire

- Lecture ou écriture de l'état du port en une seule fois
  - en lecture pas de problème particulier: vous récupérez l'état d'un ou plusieurs bits en particulier à partir de masquages

PORTB	d7	d6	d5	d4	d3	d2	d1	d0
Lignes	1	0	1	1	0	0	1	0

Var=PORTB; // ici var=0xB2

Technique des masques pour tester en lecture l'état de bits

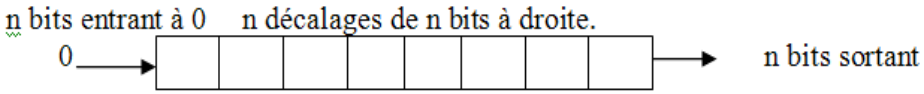
```
If (PORTB&0x21==0x21)
  { // les bits B5 et B0 sont a 1}
 If (PORTB&0x21==0x01)
 { // B5 à 0 et B0  a 1}
 If (PORTB&0x21==0x20)
 { // B5  à 1  et B0 sont a 0}
 If (PORTB&0x21==0x00)
 { // B5  à 0  et B0 sont a 0}
```

  - En écriture il faut être capable de modifier l'état des lignes que l'on souhaite sans changer les autres
    - => UTILISATION DES MASQUES

Opérateurs logiques bit à bit

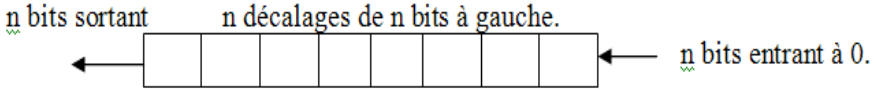
Décalages

- Décalages Logiques (nombres non signés): Déclarer les variables non Signées.
  - ✓ Décalages Logiques Droite de n bits :



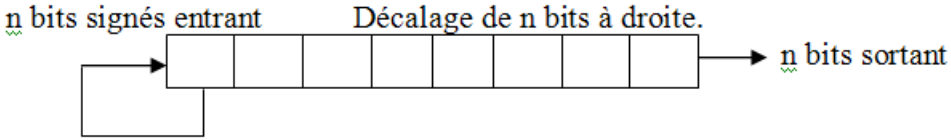
Exemple : var1 = var1 >>3 ;
avec char var1 = 0X82 avant l'opération :
Après décalage : var1 = 0X10.
Remarque : Arithmétiquement un décalage à droite équivaut à une division par 2.

  - ✓ Décalages Logiques Gauche de n bits :



Exemple : var2 = var2 << 5 ;
avec char var2 = 0X71 avant l'opération :
Après décalage : var2 = 0X 20.
Remarque : Arithmétiquement un décalage à gauche équivaut à une multiplication par2.

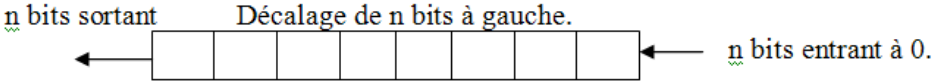
- Décalages Arithmétiques (nombres signés): Déclarer les variables Signées
  - ✓ il serait souhaitable que le bit de signe (le bit de poids le plus fort) soit conservé à 0 pour les nombres positifs, à 1 pour les nombres négatifs
  - ✓ ce type de décalage dépend du compilateur utilisé



Cas où on conserve le bit de signe à chaque décalage d'un bit à droite

Exemple : var1 = var1 >>4 ;
avec char var1 = 0X80 avant l'opération :

Après décalage : var1 = 0XF8.



Identique au décalage gauche pour un nombre non signé, puisque le bit de signe est perdu dès le premier décalage à gauche

Exemple : var2 = var2 << 2 ;
avec char var2 = 0x81 avant l'opération

Après décalage : var2 = 0X04.