Programmation des microcontrôleurs PIC en BASIC

Installation et configuration du compilateur Basic Pic

1-Installation :
Lancez SETUP.EXE, l'installation se fait automatiquement si vous acceptez les valeurs d'installation par défaut.

2-Lancement :
Dans le menu **Démarrer - Programmes** de Windows, lancez BASIC PIC.

3- Configuration :
Dans le menu **Option** de BASIC PIC entrez le répertoire d'installation qui est C:\BasicPic si vous avez acceptez les valeurs par défaut d'installation.
Ce répertoire dans **Option** est le repertoire dans lequel le compilateur ira chercher les fichiers INCLUDE.
Si le compilateur vous signale qu'il n'a pas trouvé un fichier, vérifiez que **Option** contient le chemin valide pour trouver ce fichier.

Vous pouvez maintenant essayer un premier programme.
4- Allez à **Fichier Ouvrir** et chargez le programme exemple EX1.BAS
Les exemples se trouvent dans C:\BasicPic\EXEMPES
5- Compilez par la commande **Compiler**
6- Assemblez par la commande **Assembler**
7- Cliquez sur **HEX**. Vous obtenez sur votre écran le fichier EX1.HEX produit par le compilateur.
Ce fichier EX1.HEX est le fichier qu'utilisera votre programmateur pour griller le programme EX1 dans la mémoire du microcontrôleur.Le fichier EX1.HEX est placé par le compilateur dans le même répertoire que EX1.BAS.
8- Votre microcontrôleur peut alors exécuter le programme EX1.BAS .

Ce sont les étapes 5 et 6 que vous aurez à répéter souvent lors de la mise au point de vos programmes.

Pourquoi seulement une LED et une touche ?

Les bases de la programmation en Basic ne nécessitent pas de circuits complexes, il est important de savoir que l'utilisation d'un microcontrôleur est très simple une fois le langage de programmation maîtrisé.
La connaissance du microcontrôleur que vous voulez utiliser peut être partielle.Par exemple, si vous n'utilisez pas le convertisseur analogique-digital alors vous n'avez pas à savoir comment il fonctionne ni comment le commander.Cette connaissance du µC est indépendante du langage de programmation même si elle est souvent associée à l'assembleur.Un microcontrôleur peut être comparé à un tableau de bord avec des interrupteurs de commandes et des voyants,des afficheurs ...Toutes les commandes consistent à mettre tel ou tel bit à 1 ou à 0 , à lire ou à écrire dans une adresse mémoire suivant les indications du fabricant.
C'est pourquoi la connaissance du langage de programmation est plus importante que la connaissance du microcontrôleur lui-même.
Le langage de programmation est le langage dans lequel vous écrirez vos commandes destinées à être exécutées par le µC.
L'objectif dans ce chapitre qui est une version du chapitre 1 du Manuel du BASIC orienté microcontrôleurs spécifique aux microcontrôleurs PIC de Microchip est de vous familiariser simultanément avec :

1-Le BASIC, qui en plus de sa simplicité, est de plus en plus performant et peut rivaliser dans nombreuses applications avec le langage C (évidemment si vous avez entre les mains un compilateur efficace).

2-Les microcontrôleurs PIC de Microchip

Si vous ne disposez pas de programmateur de microcontrôleurs PIC, utilisez Un microcontrôleur pré-programmé avec PICPIC.Visitez notre site web pour en savoir plus sur PICPIC.
Ce chapitre traite des bases de la programmation des microcontrôleurs PIC en BASIC orienté microcontrôleurs avec seulement une LED et une touche.
Une bonne assimilation de ce chapitre permettra d'écrire les programmes nécessaires pour les circuits électroniques les plus complexes.

Chaque microcontôleur peut être reconnu par le compilateur Basic Pic à l'aide d'un fichier d'identification à inclure au début du programme .Tous les programmes du chapitre peuvent tourner sur tous les microcontrôleurs PIC, il suffit d'inclure le fichier correspondant .

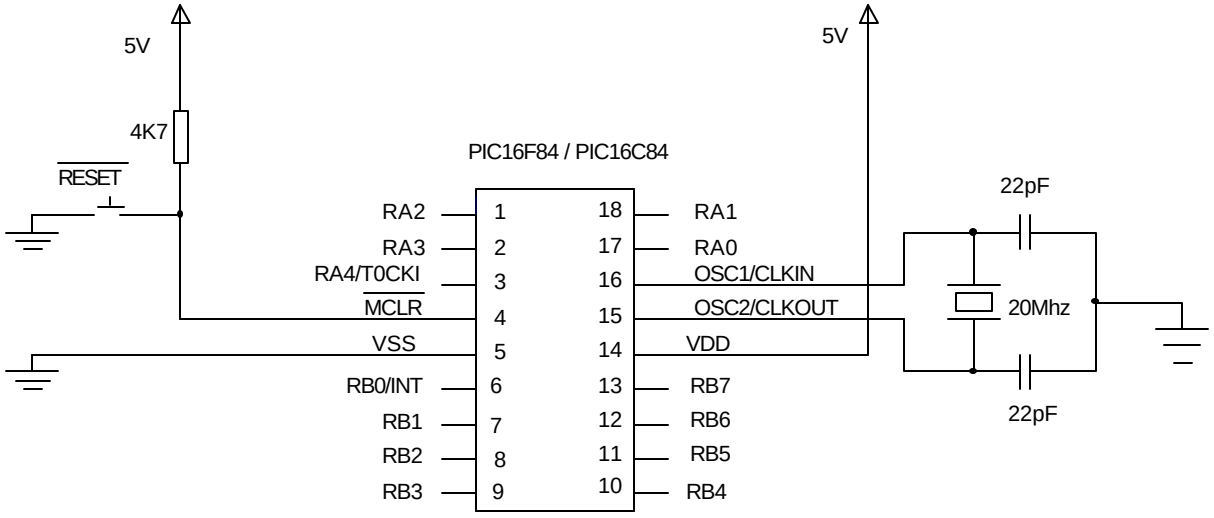
```
Include "16F84.INC"      'Programme pour PIC16F84
Include "16F876.INC"     'Programme pour PIC16F876
Include "16F877.INC"     'Programme pour PIC16F877
Include "16F876P.INC"    'Programme pour PICPIC16F876   (PICPIC)
Include "16F877P.INC"    'Programme pour PICPIC16F877   (PICPIC)
```

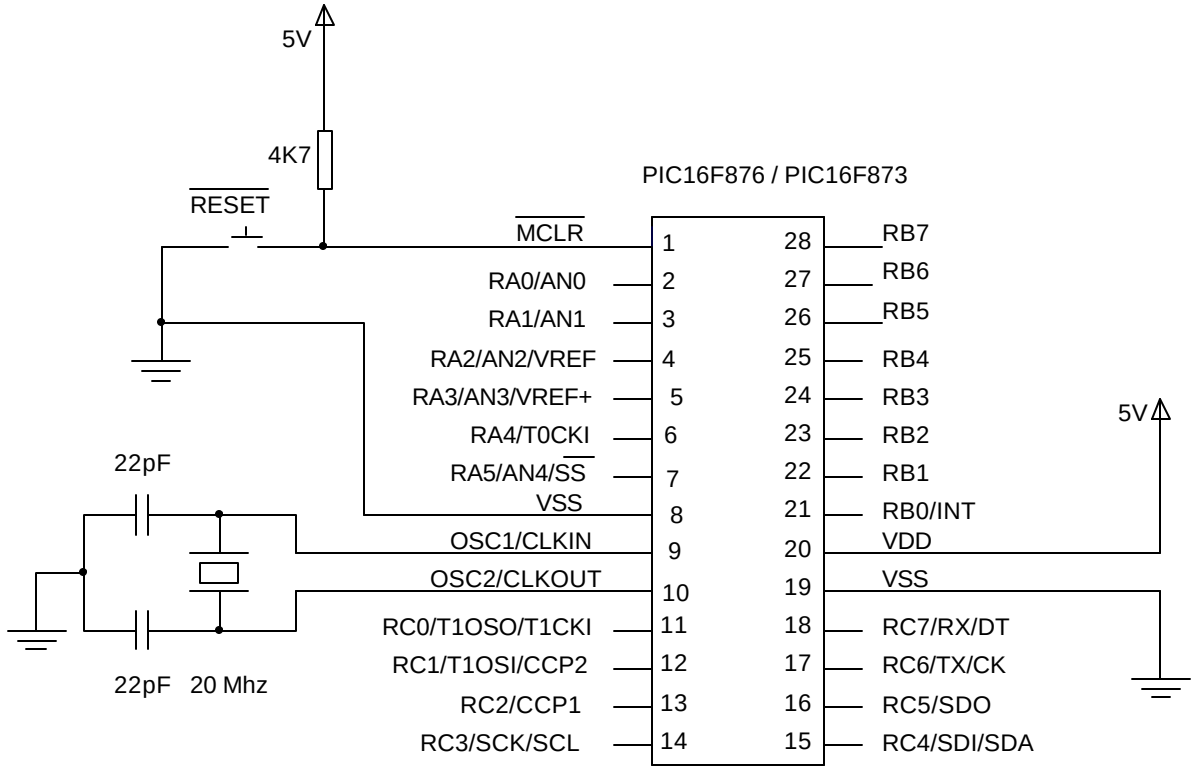
Matériel nécessaire :

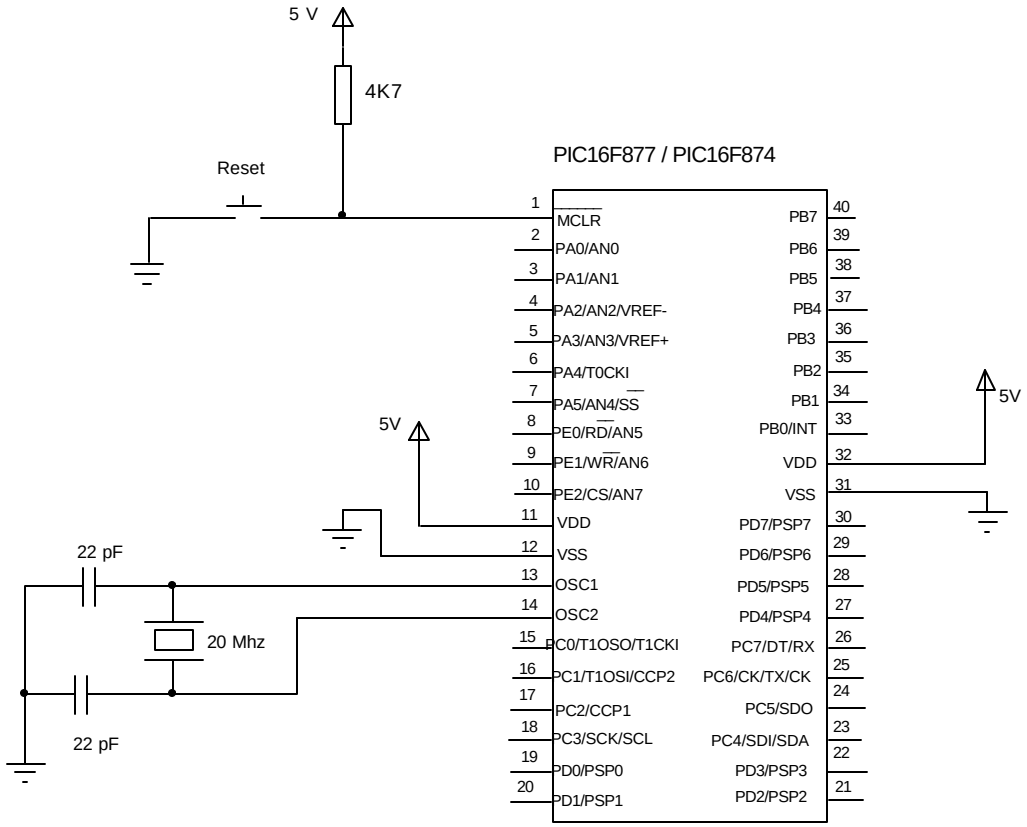
1- Un microcontrôleur PIC
2- Un programmateur à moins d'utiliser un microcontrôleur PIC auto-programmable par RS232 et pré-programmé avec PICPIC.BAS.Visitez notre site web pour plus d'informations sur PICPIC.
3- Une diode LED
4- Un poussoir

Chaque programme, une fois compilé vous donne un fichier HEX à griller dans la mémoire du microcontrôleur à l'aide de votre programmateur ou par RS232 si vous utilisez PICPIC.

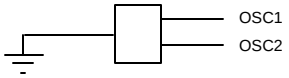
Suivant le microcontrôleur PIC que vous avez choisi, vous aurez à réaliser l'un des circuits suivants.Chaque circuit est la configuration minimale pour le fonctionnement du microcontrôleur.


5V
4K7
RESET
PIC16F84 / PIC16C84
RA2 1
RA3 2
RA4/T0CKI 3
MCLR 4
VSS 5
RB0/INT 6
RB1 7
RB2 8
RB3 9
18 RA1
17 RA0
16 OSC1/CLKIN
15 OSC2/CLKOUT
14 VDD
13 RB7
12 RB6
11 RB5
10 RB4
5V
22pF
20Mhz
22pF


5V
4K7
RESET
PIC16F876 / PIC16F873
MCLR 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF 4
RA3/AN3/VREF+ 5
RA4/T0CKI 6
RA5/AN4/SS 7
VSS 8
OSC1/CLKIN 9
OSC2/CLKOUT 10
RC0/T1OSO/T1CKI 11
RC1/T1OSI/CCP2 12
RC2/CCP1 13
RC3/SCK/SCL 14
28 RB7
27 RB6
26 RB5
25 RB4
24 RB3
23 RB2
22 RB1
21 RB0/INT
20 VDD
19 VSS
18 RC7/RX/DT
17 RC6/TX/CK
16 RC5/SDO
15 RC4/SDI/SDA
5V
22pF
22pF 20 Mhz



Si un résonnateur à la place du quartz



Partie 1 :Niveau 1

Sujets traités:

-Diriger une ligne du microcontrôleur en entrée ou en sortie
-AT
-BIT
-BYTE
-END
-FOR...TO...NEXT
-GOTO
-INCLUDE
-INPUT
-OUTPUT
-PAUSE
-REM
-STOP

Le schéma suivant, bien que simple, va nous permettre de tester nos premiers programmes.
La ligne PB4 est la ligne 4 du port B du PIC16F84, PIC16F876 , 16F877...

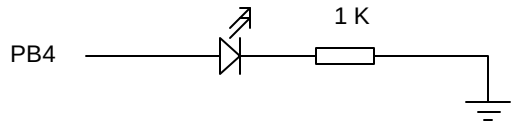


Figure 1

Quel programme permettra pour mettre à 1 la ligne PB4 ce qui aura pour effet d'allumer la LED ?
Avant de répondre à cette question , sachez qu'à chaque ligne E/S des microcontrôleurs PIC correspond un bit de direction.Ce bit permet de diriger la ligne soit en entrée soit en sortie.

Pour diriger la ligne 4 du port B en entrée, on écrira : INPUT PB4 ou bien INPUT PORTB.4
Pour diriger la ligne 4 du port B en sortie, on écrira : OUTPUT PB4 ou bien OUTPUT PORTB.4
Quand vous écrivez OUTPUT PB4 , le compilateur se charge de produire les instructions qui permettrons de mettre à 0 le bit 4 du registre TRISB.Ceci permettra de configurer la broche en sortie.
Quand vous écrivez INPUT PB4, le compilateur se charge de produire les instructions qui permettrons de mettre à 1 le bit 4 du registre TRISB. Ceci permettra de configurer la broche en entrée.
Vous pouvez écrire TRISB.4 = 0 Au lieu de OUTPUT PB4.
Vous pouvez écrire TRISB.4 = 1 Au lieu de INPUT PB4.

Pour allumer la LED , le programme doit

-Diriger la ligne PB4 en sortie ce qui correspond à l'instruction OUTPUT PB4.
-Mettre la ligne PB4 à 1 ce qui correspond à l'instruction PB4 = 1.

Ce qui nous donne le programme 1

```
    Rem Programme 1 : Allume la LED
    Include "16F876.INC"   'Programme pour PIC16F876
debut:
    Output PB4             'diriger la ligne 4 du port B en sortie
    PB4 = 1                'mettre  la ligne 4 du port B à 1
    Stop                   'passer en mode veille
End
```

REM et ' indiquent au compilateur d'ignorer la suite de la ligne.

Ce qui se trouve apès REM et ' est un commentaire pour améliorer la lisiblité du programme.
INCLUDE demande au compilateur de compiler le fichier 16F876.INC et lui indique que le programme est destiné au microcontrôleur PIC16F876

Suivant le microcontôleur, il faut rajouter une des lignes ci-dessous au début du programme.

```
include "16F84.INC"      'Programme pour PIC16F84
include "16F876.INC"     'Programme pour PIC16F876
include "16F877.INC"     'Programme pour PIC16F877
Include "16F876P.INC"    'Programme pour PICPIC16F876   (PICPIC)
Include "16F877P.INC"    'Programme pour PICPIC16F877   (PICPIC)
```

STOP va mettre le microcontrôleur en veille et sera traduite par l'instruction assembleur SLEEP.
END indique au compilateur d'arrêter la compilation.Si des lignes existent après END alors elles seront ignorées.

Pour diriger une ligne en entrée, il faut remplacer OUTPUT (sortie) par INPUT (entrée).

Avant d'aller plus loin et afin d'éviter à avoir à répéter PORTB.4 , on peut indiquer au compilateur qu'on va utiliser le symbole LED sur la ligne 4 du PORTB
Pour cela il faut écrire :

```
Bit  LED   At PORTB.4
```

A chaque fois qu'il rencontre le symbole LED ,le compilateur le remplace par PORTB.4 à la compilation.
BIT permet de déclarer une variable pouvant prendre l'une des valeurs binaires possibles qui sont 0 et 1.
Vous pouvez remplacer les symboles LED par le symbole de votre choix.
Pour allumer la LED pendant une seconde (1000 ms) il faut

- -Diriger la ligne en sortie
- -Allumer la LED
- -Attendre 1 seconde
- -L'éteindre

Ce qui se traduit par le programme suivant

```
     'Programme 2 : Allume la LED pendant 1 seconde
     Include "16F876.INC"  'Programme pour PIC16F876
     Bit LED  At PORTB.4   'Une LED est connectée sur la ligne PB4
debut:
     Output LED            'dirige la ligne qui commande la LED en sortie
     LED  = 1              'allumer LED.Vous pouvez écrire High LED
     Pause 1000            'attendre 1000 ms
     LED  = 0              'éteindre la LED.Vous pouvez écrire Low LED
     Stop
End
```

L'instruction GOTO va nous permettre de modifier le programme précédent pour clignoter la LED suivant une boucle infinie.

```
     'Programme 3 : Clignote la LED
     Include "16F876.INC"  'Programme pour PIC16F876
     Bit LED  At PORTB.4   'Une LED est connectée sur la ligne 4 du port B

debut:
     Output LED            'dirige la ligne qui commande la LED en sortie
boucle:
     LED = 1               'allumer LED.Vous pouvez écrire High LED
     Pause  500            'attendre 0.5 s
     LED = 0               'éteindre LED
     Pause 500             'attendre 0.5 s
     Goto boucle           'aller à l'instruction située au label boucle
End
```

Vous remarquerez que dans le programme , l'instruction STOP a été supprimée car le programme est une boucle infinie donc le microcontrôleur ne doit pas s'arrêter.

Les habitués des microcontrôleurs PIC ne doivent pas s'étonner de trouver dans les exemples DDRB et non TRISB pour nommer le registre de direction du port B, en effet le compilateur Basic Pic accepte l'une ou l'autre des nominations.Vous pouvez même choisir un autre symbole quelconque pour désigner les registres du microcontrôleur.

La boucle décrite dans l'exemple précédent est une boucle infinie.Dans la pratique on n'aura pas besoin de clignoter indéfiniment une LED.Par contre on peut avoir besoin de clignoter une LED un nombre fini de fois. Pour cela on fait appel à une instruction très utilisée en langage évolué et qui prend différentes formes suivant le langage.Il s'agit de l'instruction FOR ...TO ... NEXT.
La fonction principale de l'instruction FOR est l'exécution d'une séquence d'instructions un nombre de fois prédéterminé.
Pour exécuter 10 fois une séquence d'instructions

-On choisit un compteur initialisé avec la valeur 1
-On fixe la limite que doit atteindre le compteur à 10
Si le compteur dépasse cette limite alors l'exécution de la séquence est arrêtée

Reprenons l'exemple précédent et faisons clignoter la LED 10 fois.

```
     'Programme 4 : Clignote la LED 10 fois
     Include "16F876.INC"  'Programme pour PIC16F876
     Bit  LED  At PORTB.4  'Une LED est connectée sur la ligne 4 du port B
     Byte p                'p est le compteur de la boucle FOR
debut:
     Output LED            'dirige la ligne qui commande la LED en sortie
     For p=1 TO 10         'quitte la boucle si p dépasse 10
        LED = 1
        Pause  500
        LED = 0
        Pause 500
     Next                  'augmente p de 1 et reprend la boucle FOR
     Stop                  'ici quand p prend la valeur 11
End
```

BYTE permet de déclarer une variable pouvant prendre toute valeur comprise entre 0 et 255.
Déclarer une variable permet de lui réserver une adresse dans la mémoire du microcontrôleur.
Une boucle FOR peut être traduite par les 4 étapes suivantes:

1- Charger le compteur avec une valeur initiale.Ici la valeur 1
2- Le compteur est comparé à la valeur limite.Ici la valeur 10.
Si cette limite n'est pas dépassée alors le programme passe à l'étape 3.
Sinon, le programme exécute l'instruction qui suit NEXT.
3-Exécution des instructions situées entre les lignes FOR et NEXT
4- Le compteur est augmenté de 1 (p = p+1), puis le programme passe à l'étape 2.

A chaque itération, le compteur est augmenté de 1.On dit que le pas de la boucle est 1.
Il est possible d'augmenter le compteur d'une valeur autre que 1, dans ce cas on utilisera la
la syntaxe suivante :

```
FOR ... TO ... STEP ...
   ...
   ...
   ...
 NEXT
```

D'autres types de variables sont disponibles :

WORD	sur 16 bits, pour les données entières de 0 à 65535
FLOAT	sur 32 bits,pour les données numériques à virgule et pour les fonctions mathématiques
STRING	sur 1 à 255 octets pour les chaînes de caractères (texte)
DIM	pour les tableaux.

Partie 2 :Niveau 2

Sujets traités:

-Les sous-programmes :GOSUB RETURN
-IF THEN
-IF THEN ELSE
-IF END IF
-DO WHILE LOOP
-DO LOOP WHILE
-WHILE WEND

Dans cette partie on va rajouter à notre montage de la figure 1, une touche qui va permettre d'envoyer des commandes au microcontôleur.

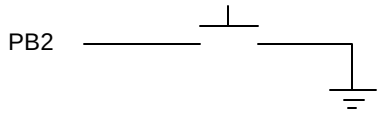


Figure 2

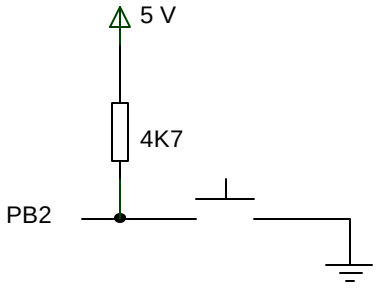


Figure 3

La ligne 2 du port B sur laquelle est connectée la touche doit être orientée en entrée.Ceci se fait à l'aide de l'instruction INPUT PB2 et maintenu à un niveau haut , soit par la résistance de rappel interne du microcontôleur (figure 2) , soit par une résistance externe (figure 3).
A chaque appui sur la touche, la ligne passe au niveau zéro .Un test de l'appui sur la touche se traduit donc par un test du niveau logique de la broche correspondante, il faut donc tester le bit PORTB.2 .
On peut rajouter le symbole PINB à la même adresse que PORTB pour les microcontrôleurs PIC pour désigner les broches du port B.
Ceci est fait dans le fichier d'identification du microcontrôleur ,soit 16F876.INC pour PIC16F876.

```
Byte PINB At 0x06
```

Cette ligne permet de déclarer la variable PINB et de préciser son adresse ici l'adresse est 6.
C'est un cas de figure où l'adresse n'est pas calculée par le compilateur mais imposée par le programmeur.
Rappelez-vous qu'il est toujours possible d'utiliser deux symboles ou plus pour identifier une même zone mémoire.

Pour vérifier que la touche a été prise en compte par le programme, nous allons allumer la LED pendant 0.5 s à chaque appui.
Ceci nous donne le programme 5 :

```
     'Programme 5 : Allume la LED à chaque appui sur la touche
     Include "16F876.INC"   'Programme pour PIC16F876
     Bit  LED  At PORTB.4   'Une LED est connectée sur la ligne 4 du port B
debut:
     Output LED             'configure la ligne 4 du port B en sortie
     Input PB2              'configure la ligne 2 du port B en entrée
     PULLUPBit = PULLUP     'ramener les lignes du port B à 5 V (fig 2)
boucle:
     If PB2 = 0 Then LED = 1 : Pause 500 : LED = 0
     Goto boucle
END
```

Si on veut clignoter la LED à haute vitesse en absence d'un appui sur la touche ,on écrira

```
   IF PB2 = 0 THEN LED = 1 : PAUSE 500 : LED = 0 ELSE LED = 1 :LED  =0
```

Si PB2 = 0 alors clignoter avec une pause de 500 ms , sinon clignoter sans pause

Si le nombre d'instructions à exécuter dans le cas où le test est réussit est trop important pour tenir sur une seule ligne alors il faut faire appel à la deuxième forme de l'instruction IF

```
     IF PB2=0 THEN
        LED = 1
        PAUSE 500
        LED = 0
        '...
        '...
        '...
     END IF
```

Ou bien en utilisant ELSE

```
     IF PB2=0 THEN
        LED = 1
        PAUSE 500
        LED = 0
        '...
        '...
        '...
     ELSE
        '...
        '...
        '...
     END IF
```

Si on voulait compter le nombre d'appuis sur la touche et le signaler en clignotant la LED, on pourrait faire appel à un sous-programme qui clignote la LED suivant le contenu d'une variable qui indique le nombre d'appuis sur la touche.

```
     'Programme 6 :Compte le nombre d'appuis sur la touche
      Include "16F876.INC"  'Programme pour PIC16F876
      Bit  LED  At PORTB.4  'Une LED est connectée sur la ligne PB4
      Byte p                'pour stocker le nombre d'appuis sur la touche
      Byte i                'compteur utilisé dans le sous-programme clignote

debut:
      Output LED            'configure la ligne 4 du port B en sortie
      Input PB2             'configure la ligne 2 du port B en entrée
      PULLUPBit = PULLUP    'ramener les lignes du port B à 5 V
      p = 0                 'au départ,le nombre d'appuis sur la touche est nul

boucle:
      If PB2 = 0 Then p=p+1:Gosub clignote
      goto boucle

clignote:                   'ici commence le sous-programme clignote
      For i = 1 TO p        'Exécute p fois la séquence qui suit
         LED = 1
         Pause 1000         '1000 ms
         LED = 0
         Pause 1000
      Next
      Return                'Retourne à l'instruction qui suit GOSUB clignote
End
```

S'il ya appui sur sur la touche (IF PB2 = 0) , le programme :

- augmente p de 1 (p=p+1)
- Fait un saut au sous-programme pour clignoter la LED p fois (GOSUB)
- revient à l'instruction qui suit l'instruction GOSUB (RETURN)

L'adresse à laquelle doit revenir le programme quand il rencontre l'instruction RETURN a été sauvegardée sur la pile du microcontrôleur.
Du fait que l'adresse de retour a été sauvegardée sur la pile, le nombre d'appels imbriqués de sous-programmes dépend de la taille de la pile du microcontrôleur..

Si on veut que la LED clignote tant qu'il y a un appui sur la touche, on fera appel à une instruction de répétition

```
DO WHILE  PB2=0             'une boucle à répéter tant que PINB.2=0
    LED = 1
    PAUSE 500
    LED = 0
    PAUSE 500
LOOP                        'fin de la boucle DO, la répéter si PINB.2=0
```

ou bien

```
WHILE  PB2=0                'une boucle à répéter tant que PINB.2=0
    LED = 1
    PAUSE 500
    LED = 0
    PAUSE 500
WEND                        'fin de la boucle WHILE, la répéter si PINB.2=0
```

Ce qui donne le programme 7

```
     'Programme 7 : Clignote la LED tant que la touche est appuyée
     Include "16F876.INC"  'Programme pour PIC16F876
     Bit  LED  At PORTB.4
debut:
     Output LED            'configure la ligne 4 du port B en sortie
     Input PB2             'configure la ligne 2 du port B en entrée
     PULLUPBit = PULLUP    'ramener les lignes du port B à 5 V

boucle:
     If PB2=0 Then Gosub clignote
     Goto boucle

clignote:                  'ici commence le sous-programme clignote
     Do While  PB2=0       'une boucle à répéter tant que PINB.2=0
       LED = 1
       PAUSE 500
       LED = 0
       PAUSE 500
     Loop                  'fin de la boucle DO, la répéter si PINB.2=0
     Return                'fin du sous-programme.
End
```

Dans le programme précédent, la LED ne clignote que s'il y a appui sur une touche, donc la boucle n'est entamée que si la ligne PINB.2 passe à zéro.
L'instruction DO...LOOP WHILE... permet d'entamer une boucle et de ne tester une condition qu'à la suite d'une première exécution de cette boucle.
Elle peut être utilisée si l'on veut clignoter la LED et arrêter le clignotement dès qu'il y'a appui sur la touche.
Ceci est fait par le programme 8 :

```
     'Programme 8 : Clignote la LED tant que la touche n'est pas appuyée

     Include "16F876.INC"  'Programme pour PIC16F876
     Bit  LED  At PORTB.4
debut:
     Output LED            'configure la ligne 4 du port B en sortie
     Input PB2             'configure la ligne 2 du port B en entrée
     PULLUPBit = PULLUP    'ramener les lignes du port B à 5 V
boucle:
     Do                    'exécuter la boucle qui suit
       LED = 1
       Pause 500
       LED = 0
       Pause 500
     Loop While PB2=1      'recommencer tant que PB2=1 (pas d'appui sur la touche)
     Stop                  'ici si la broche PB2 passe à zéro(appui sur la touche)
End
```

Voici les 2 formes possibles de l'instruction DO

```
     DO
       ...
       ...
       ...

     LOOP WHILE ...
```

```
DO WHILE ...
  ...
  ...
  ...

LOOP
```

Partie 3 :Les afficheurs LCD - la commande INCLUDE - les variables de type BIT

Le programme 9 illustre la facilité avec laquelle un afficheur LCD peut être géré avec Basic Pic.
Un module LCD.BAS se charge de la gestion de l'afficheur LCD.

```
'Programme 9 :Utilisation d'un afficheur LCD

Include "16F84.INC"             'Programme pour PIC16F84
Include "lcd.bas"               'utilise un afficheur LCD
debut:
     LcdInit                    'Initialise l'afficheur
boucle:
     Print "essai"              'ou bien  PRINT "Digimok"
     Stop
End
```

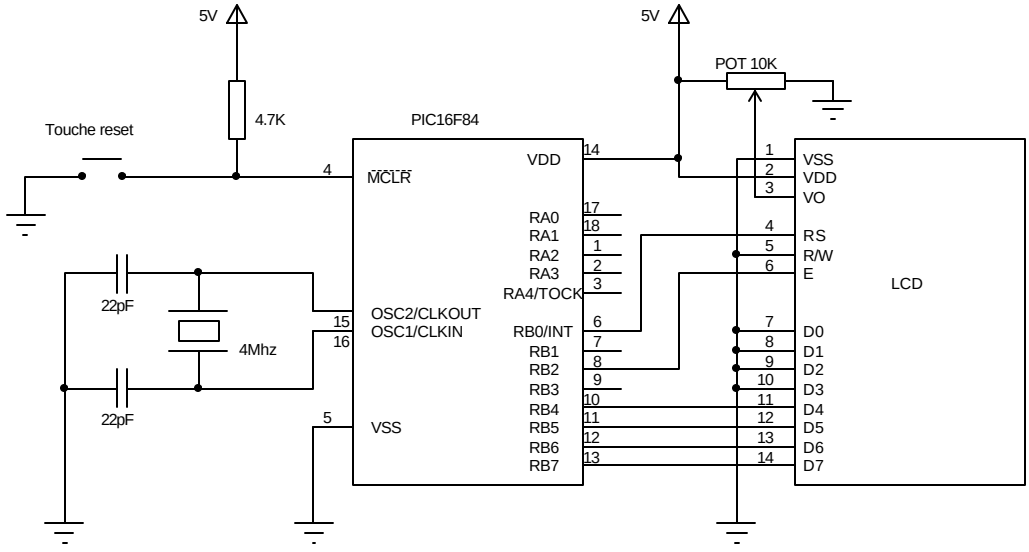


Fig 5

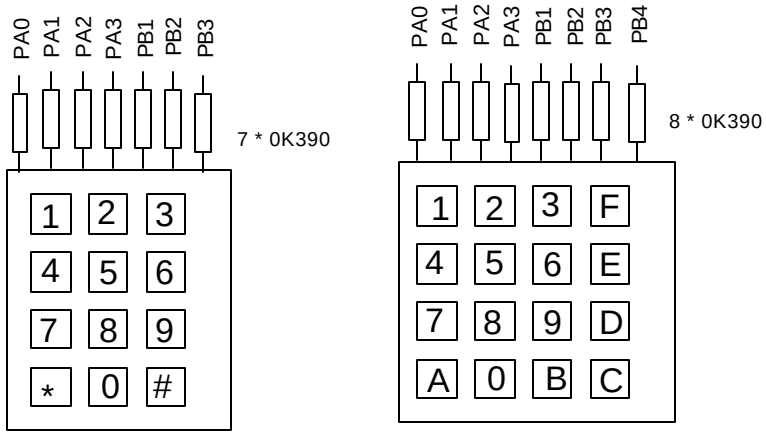
La question qui se pose est : Que dois-je modifier dans le programme si l'afficheur LCD n'est pas connecté suivant le shéma ci-dessus ?
L'utilisation d'un afficheur LCD illustre la facilité avec laquelle Basic Pic permet de gérer les périphériques grâce à la programmation modulaire et grâce aux manipulations aux niveaux des variables de type BIT.

Jusqu'à maintenant INCLUDE nous a servi de sélectionner le microcontrôleur,ici INCLUDE permet de connecter à votre programme le module LCD.BAS de manière analogue à la connection matérielle de l'afficheur LCD sur votre circuit.
Si l'afficheur LCD est connecté autrement, vous n'avez pas à réécrire le module LCD.BAS , vous aurez à modifier seulement les connections en utilisant les variables BIT.

```
byte  LCD_DATA  at   PORTB          'LCD_DATA seulement poids fort du port B
byte  LCD_DDR   at   TRISB
bit   LCD_RS         PORTB.0
bit   LCD_E          PORTB.2
```

Il suffit donc de modifier dans le module LCD.BAS les 4 lignes ci-dessus pour les faire correspondre avec votre circuit.

Partie 4 :Les claviers matriciels - Les fonctions FUNCTION



Ci-dessus vous avez la connection possible d'un clavier matriciel 12 touches et d'un clavier matriciel 16 touches.Les broches auquelles est connecté le clavier n'ont aucune importance dans le module qui gère le clavier, le clavier peut être connecté différement, il suffit de modifier les connections logicielles par les variables BIT.
Le module 12TOUCHE.BAS comprend la configuration du clavier à modifier si la connection du clavier sur votre circuit est différente.

```
Bit  _L1    At   PORTA.0
Bit  _L2    At   PORTA.1
Bit  _L3    At   PORTA.2
Bit  _L4    At   PORTA.3
Bit  _C1    At   PINB.1
Bit  _C2    At   PINB.2
Bit  _C3    At   PINB.3
```

Adaptez les lignes précédentes à votre configuration et essayez le programme suivant :

```
'Programme 10 :Lit un clavier matriciel et envoie au PC la touche appuyée
'Version fonction
Include "16F876.INC"              'Programme pour PIC16F876
Include "12TOUCHE.BAS"            'Utilise un clavier matriciel 12 touches
Include "UARTPIC.BAS"             'Utilise liaison série pour Debug

debut:
     Serinit _19200_BAUD          'Initialise la liaison Série
boucle:
     a=LireClavier()              'Lit le clavier
     If a<> 0 Then Seroutcar a    'a est <>0 si une touche est appuyée
     Goto boucle
End
```

Analysons le module 12TOUCHE.BAS pour comprendre la manière avec laquelle le programme scrute le clavier :

```
'module 12touche.bas
'Enlevez la ligne   PCFG2=1 : PCFG1=1  si non PIC16F87X
Declare Function LireClavier()  As  Byte
Bit  _L1    At   PORTA.0
Bit  _L2    At   PORTA.1
Bit  _L3    At   PORTA.2
Bit  _L4    At   PORTA.3
Bit  _C1    At   PINB.1
Bit  _C2    At   PINB.2
Bit  _C3    At   PINB.3
Function LireClavier()  as  Byte
    'PCFG0=1 : PCFG1=1              'Port A digital - PIC16C715
     PCFG2=1 : PCFG1=1              'Port A digital - PIC16F87X

     PullUpBit=PULLUP               'PORTB pull-ups ON - tout PIC16

     pause 500                      'ms

     Input  _C1                     'colonnes en entrée  et à 5 V par pull-up
     Input  _C2
     Input  _C3

     Output _L1                     'lignes en sortie
     Output _L2
     Output _L3
     Output _L4

     _L4=1:_L3=1:_L2=1:_L1=0
     Nop                             'sinon '1' ignoré
     If _C1= 0 Then Return('1')
     If _C2= 0 Then Return('2')
     If _C3= 0 Then Return('3')

     _L4=1:_L3=1:_L2=0:_L1=1
     If _C1= 0 Then Return('4')
     If _C2= 0 Then Return('5')
     If _C3= 0 Then Return('6')

     _L4=1:_L3=0:_L2=1:_L1=1
     If _C1= 0 Then Return('7')
     If _C2= 0 Then Return('8')
     If _C3= 0 Then Return('9')

     _L4=0:_L3=1:_L2=1:_L1=1
     If _C1= 0 Then Return('*')
     If _C2= 0 Then Return('0')
     If _C3= 0 Then Return('#')
     Return(0)                       'pas de touche
     End Function
```

Analysons le module 12touche.bas :

1 - Comme toute routine (fonctions ou procédure) , la fonction LireClavier doit être déclarée .

```
Declare Function LireClavier()  As  Byte
```

Cette fonction est en fait un sous-programme qui permet de retourner un octet (le type de la fonction est Byte) qui représente le code ASCII de la touche appuyée.

2- Les 4 lignes du clavier sont connectées aux broches PA0,PA1,PA2 et PA3

```
Bit  _L1    At   PORTA.0
Bit  _L2    At   PORTA.1
Bit  _L3    At   PORTA.2
Bit  _L4    At   PORTA.3
```

3- Les 3 colonnes du clavier sont connectées aux broches PB1,PB2 et PB3

```
Bit  _C1    At   PINB.1
Bit  _C2    At   PINB.2
Bit  _C3    At   PINB.3
```

Ces colonnes seront ramenées à 5 V par les résistances de rappel interne du microcontrôleur.C'est pour cette raison que le port B a été choisi.Sinon, il faut les ramener à 5 V par des résistances externes.

4- Le port A est configuré en port digital dans le cas de microcontrôleur PIC avec ADC.

```
     PCFG2=1 : PCFG1=1              'Port A digital - PIC16F87X
```

5- Les lignes du port B et donc les colonnes du clavier sont ramenées à 5 V par les résistances internes

```
     PullUpBit=PULLUP               'PORTB pull-ups ON - tout PIC16
```

6- Les colonnes sont configurées en entrée et les lignes en sortie

```
     Input  _C1                     'colonnes en entrée  et à 5 V par pull-up
     Input  _C2
     Input  _C3

     Output _L1                     'lignes en sortie
     Output _L2
     Output _L3
     Output _L4
```

7- Maintenant peut commencer la scrutation du clavier.

Le programme met une seule ligne à la fois au niveau 0 et teste les broches qui sont en entrée (PORTB). S'il trouve une broche au niveau 0 alors qu'elle est à 5 V au repos, cela permet de déduire qu'une touche est appuyée. L'appui sur une touche produit une liaison entre une broche du port A (les lignes) et une broche du port B (les colonnes).Si aucune broche du port B n'est au niveau 0, le programme refait la même opération en mettant la ligne suivante (broche suivante du port A) à zéro et puis en testant les broches du port B. Le numéro de la ligne et le numéro de la colonne permet de déduire quelle touche a été appuyée.

Par exemple,si le programme met la broche reliée à ligne 2 à zéro et trouve la broche reliée à la colonne 1 au niveau 0, alors il en déduit que c'est la touche '4'

Voici ci-dessous LireClavier sous la forme d'un sous-programme et non une fonction :Pour l'appeler, il faut écrire `Gosub LireClavier` et vous obtenez le code ASCII de la touche dans la variable `a`

```
'Enlevez la ligne   PCFG2=1 : PCFG1=1  si non PIC16F87X
Bit  _L1    At   PORTA.0
Bit  _L2    At   PORTA.1
Bit  _L3    At   PORTA.2
Bit  _L4    At   PORTA.3
Bit  _C1    At   PINB.1
Bit  _C2    At   PINB.2
Bit  _C3    At   PINB.3

LireClavier:
    'PCFG0=1 : PCFG1=1              'Port A digital - PIC16C715
     PCFG2=1 : PCFG1=1              'Port A digital - PIC16F87X

     PullUpBit=PULLUP               'PORTB pull-ups ON - tout PIC16

     pause 500                      'ms

     Input  _C1                     'colonnes en entrée  et à 5 V par pull-up
     Input  _C2
     Input  _C3

     Output _L1                     'lignes en sortie
     Output _L2
     Output _L3
     Output _L4

     _L4=1:_L3=1:_L2=1:_L1=0
     Nop                                'sinon '1' ignoré
     If _C1= 0 Then a = '1' : Return
     If _C2= 0 Then a = '2' : Return
     If _C3= 0 Then a = '3' : Return

     _L4=1:_L3=1:_L2=0:_L1=1
     If _C1= 0 Then a = '4' : Return
     If _C2= 0 Then a = '5' : Return
     If _C3= 0 Then a = '6' : Return

     _L4=1:_L3=0:_L2=1:_L1=1
     If _C1= 0 Then a = '7' : Return
     If _C2= 0 Then a = '8' : Return
     If _C3= 0 Then a = '9' : Return

     _L4=0:_L3=1:_L2=1:_L1=1
     If _C1= 0 Then a = '*' : Return
     If _C2= 0 Then a = '0' : Return
     If _C3= 0 Then a = '#' : Return
     a = 0                               'pas de touche
     Return
```

Le programme 10 devient alors :

```
'Programme 11 :Lit un clavier matriciel et envoie au PC la touche appuyée
'Version sous-programme
Include "16F876.INC"              'Programme pour PIC16F876
Include "12TOUCHE.BAS"            'Utilise un clavier matriciel 12 touches
Include "UARTPIC.BAS"             'Utilise liaison série pour Debug
debut:
     Serinit _19200_BAUD          'Initialise la liaison Série
boucle:
     Gosub LireClavier            'Lit le clavier
     If a<> 0 Then Seroutcar a    'a est <>0 si une touche est appuyée
     Goto boucle
End
```

Partie 5 :Testez-vous

Voici la listes des programmes décrits dans ce chapitre, si vous pouvez les réécrire, vous avez alors les bases nécessaires pour vous lancer dans des programmes complexes
Votre programme commencera par :

```
Include "16F876.INC"              'Programme pour PIC16F876
ou
Include "16F84.INC"               'Programme pour PIC16F84
```

Programme 1 : Allume une LED connectée à la ligne 4 du port B.
Programme 2 : Allume la LED pendant 1 seconde.
Programme 3 : Clignote la LED.
Programme 4 : Clignote 10 fois la LED.
Programme 5 : Allume la LED à chaque appui sur un poussoir connecté à la ligne 2 du port B.
Programme 6 : Compte et affiche par la LED le nombre d'appuis sur le poussoir.
Utilisez un sous-programme.
Programme 7 : Clignote la LED tant que le poussoir est appuyé.
Programme 8 : Clignote la LED tant que le poussoir n'est pas appuyé.