OBJECTIFS :

1-S'initier à la programmation en langage C-CCS pour PIC .

2-Développer des applications embarquées sur microcontrôleurs.

3- Programmer le microcontrôleur PIC et valider le fonctionnement sur le kit de développement.

Programmation et compilateur :

CCS C est un compilateur très largement utilisé pour PIC. Il offre la programmation de presque tous les microcontrôleurs PIC, ainsi que son interface attrayante et conviviale.

INITIATION :

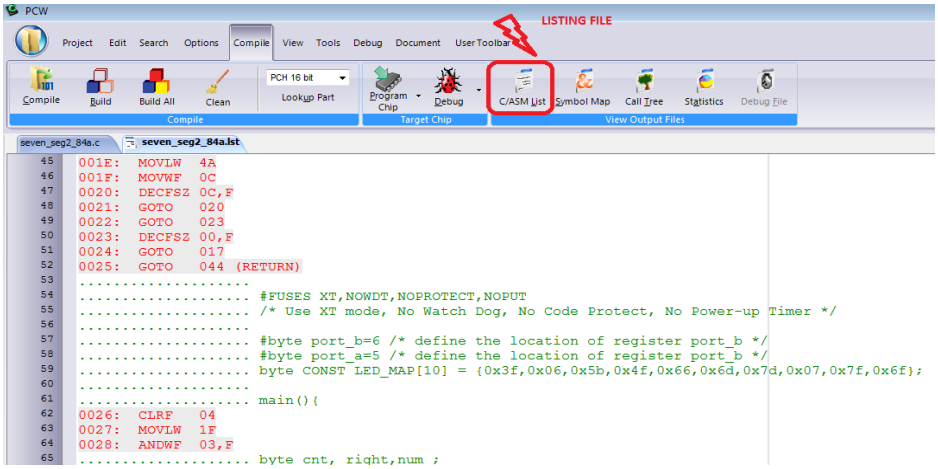
Étape 1 : Nous écrirons un programme qui fera clignoter une LED avec un délai de 1 seconde.

- Lancez le compilateur CCS C et créez un nouveau fichier.
- Enregistrez le fichier à l'emplacement souhaité.
- Copiez le code suivant dans le compilateur

```
#include < 16 F84.h>
#fuses HS,WDT,NOPROTECT
#use delay(clock= 4000000 )

void main () {
while (1) {
output_toggle(PIN_B0) ;
delay_ms( 200 ); }}
```

- Appuyez sur 'F9′ pour compiler le code. Il enregistrera le fichier HEX avec le même nom que le code. Nous utiliserons le fichier .HEX pour le télécharger dans notre microcontrôleur.



C/ASM List

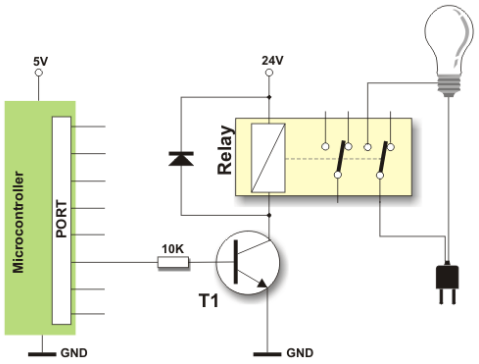
Opens the listing file in read only mode. The file must have been compiled to view the list file. If open, this file will be updated after each compile. The listing file shows each C source line and the associated assembly code generated for the line.

-Examiner aussi les fichiers à extension .asm, .hex , .err .Que contiennent ils ? Ils servent à quoi ?

<u>Étape 2</u>-Dans cette étape, vous allez saisir le schéma du montage à base de pic pour le simuler sous Proteus/Isis. Charger le fichier HEX généré par le compilateur à l'étape 1 dans le PIC. Maintenant, tout est prêt. Appuyez sur « F12 » pour démarrer la simulation. La LED commencera à clignoter normalement.

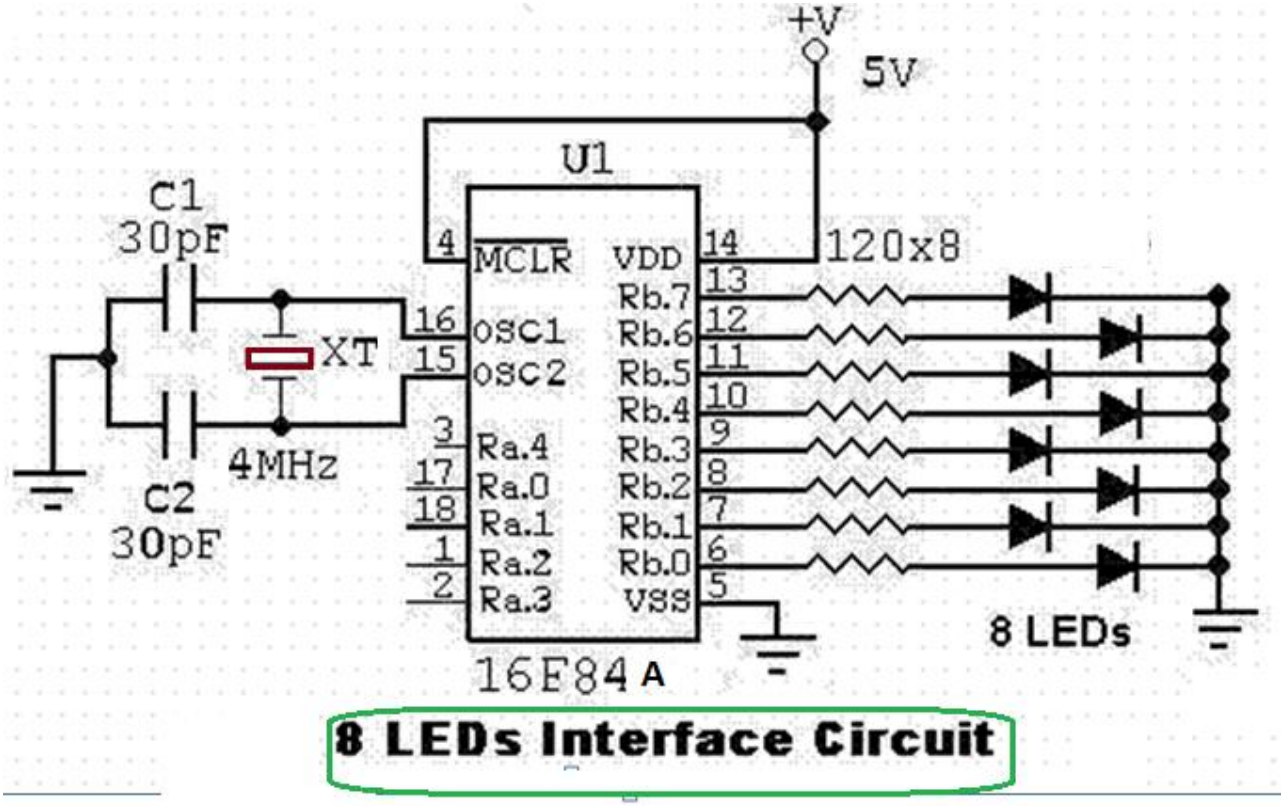
-Vérifier à l'oscilloscope la période du signal carré sur la broche RB0.vous devez avoir 400ms.

-Modifier le programme en ajoutant un bouton poussoir sur RA0 pour **marche /arrêt** du télé-rupteur utilisé en éclairage des escaliers d'un immeuble de 03 étages. Vérifier par simulation son bon fonctionnement.



<u>PREMIERE PARTIE</u>

Étant donné que n'importe quelle broche PORTB est capable de générer un max. de 20mA par broche avec PORTB capable de fournir un courant total maximum de 100mA, nous pouvons donc utiliser PORTB pour piloter directement l'affichage LED. Comme indiqué dans le circuit, 8 LED sont connectées au PORTB avec chaque résistance de limitation de courant de 120 Ohm.



Le code suivant, led.c montre comment allumer et éteindre les LEDs. Le programme fait clignoter la série de LED de LED1 à LED8 s'allumant et s'éteignant les unes après les autres de manière séquentielle. Lorsqu'on atteint LED8, il répète le processus à nouveau en commençant à LED1.

Example to drive 8 LEDS

```
/*********************************************
*    WHILE loop construct    *
 *    to drive 8 LEDS connected to port B      *
  *********************************************/
#include <16F84A.h>

#USE DELAY( CLOCK=4000000 ) /* Using a 4 Mhz clock */

#FUSES XT,NOWDT,NOPROTECT,NOPUT
/* Use XT mode, No Watch Dog, No Code Protect, No Power-up Timer */
#byte port_b=6 /* define the location of register port_b */
 main(){
 byte cnt; value;
   set_tris_b(0); /* set port_b to be outputs */
   port_b = 0; /* initialize All port_b outp/uts to be zero */
   value = 0x01;
    while( TRUE )
    { /* forever loop using WHILE construct */
     cnt = 0;
   while ( cnt<8 )
     {
      port_b = value;
       DELAY_MS(1000);
        value = value << 1; /* shift left will put 0x01, 0x02, 0x04, 0x08, 0x10 */
          cnt++; /* 0x20, 0x40, 0x80 to port_b */
         }
       }
     }
```

TACHE 1 :

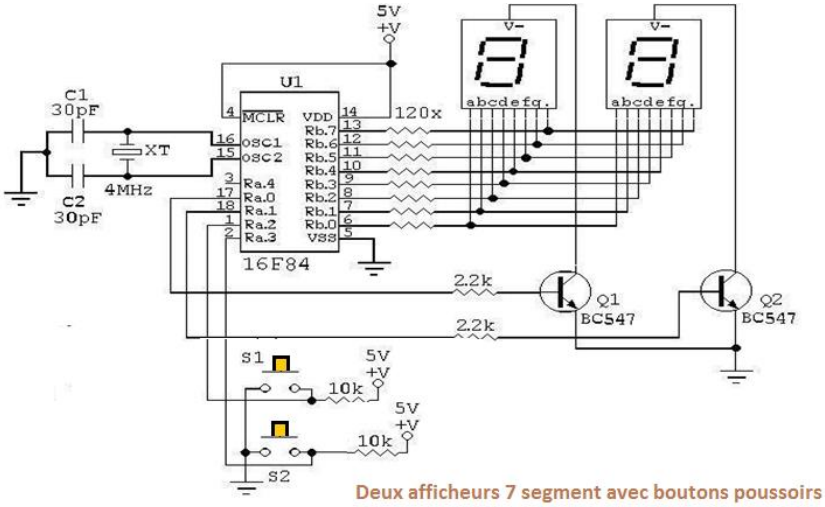
Modifiez le programme ci-dessus pour afficher la séquence suivante :

LED1 et LED8 s'allument, délai de 1 s ;
LED2 et LED7 s'allument, délai de 1 s ;
LED3 et LED6 s'allument, délai de 1 s ;
LED4 et LED5 s'allument, délai de 1 s ;
Rebouclez et répétez à nouveau pour toujours.

DEUXIEME E PARTIE :

DEUX afficheurs à 7 SEGMENTS & 2 boutons poussoirs

Étant donné que nous pouvons fournir un maximum de 20 mA par broche pour PORTB et que le courant de source maximal total de PORTB est de 100 mA, nous pouvons donc utiliser PORTB pour piloter directement l'affichage LED. Comme indiqué dans le circuit, 2 LED à cathode commune à 7 segments sont connectées au PORTB avec une résistance de limitation de courant de 120 Ohms. Chaque segment de deux LED est lié en parallèle à P1. Q1 et Q2 sont activés par le haut logique de RA0 et RA1 qui commutent à la masse sur les broches de cathode communes respectives. Les broches RA2 et RA3 sont utilisées pour lire 2 boutons poussoirs S1 ou S2.



Deux afficheurs 7 segment avec boutons poussoirs

Le code suivant, 7-SEG.C montre comment piloter deux afficheurs à 7 segments. Le programme affiche d'abord les chiffres sur l' afficheur de gauche et compte de 0 à 9 avec un délai de 1 seconde entre chaque chiffre. Ensuite, il passe à l'afficheur de droite et répète les mêmes tâches, puis toute la séquence est répétée depuis le début.

```
/****************************************
 * two 7-Segment LEDs
****************************************/
#include <16F84A.h>
#USE DELAY( CLOCK=4000000 ) /* Using a 4 Mhz clock */
#FUSES XT,NOWDT,NOPROTECT,NOPUT
/* Use XT mode, No Watch Dog, No Code Protect, No Power-up Timer */
#byte port_b=6              /* define the location of register port_b */
#byte port_a=5                 /* define the location of register port_b */
byte CONST LED_SEGM[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
main(){
byte cnt, right,num ;
set_tris_b(0); /* set port_b as outputs */
 set_tris_a(0); /* set port_a as output */
  port_b = 0; /* ZERO port_a & port_b */
  port_a = 0;

for( ;; ){
 for (right=1;right<3;right++){
   port_a = right;
    for (cnt=0;cnt<10;cnt++){
     port_b = LED_ SEGM [cnt];
       DELAY_MS(1000); /* one second delay */

          }   }    } }
```

TACHE1 :

Les deux LED à 7 segments sont maintenant conçues pour afficher 00 simultanément et commencer à compter automatiquement en incrémentant de un avec un délai d'une seconde entre chaque incrément. Lorsque le nombre atteint 99, il doit se réinitialiser à 00 et recommencer

TACHE2 :

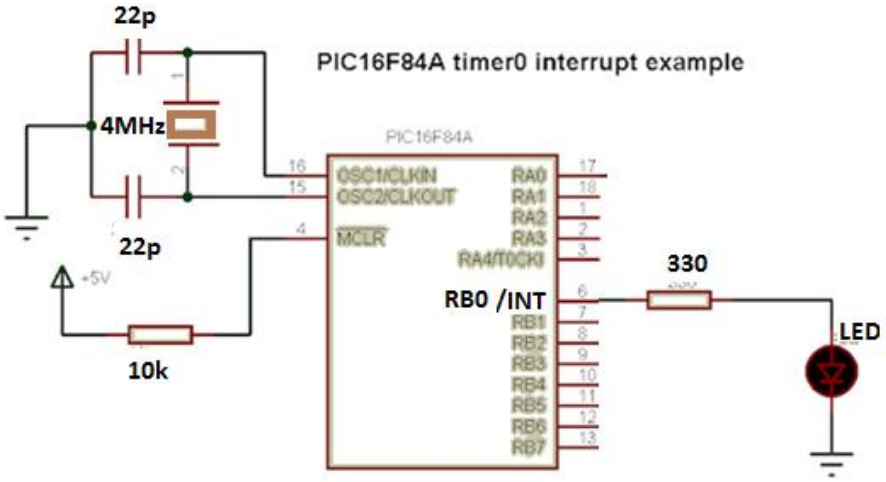
Il s'agit d'une modification de l'affectation 1 où l'incrément de un est exécuté uniquement lorsque le bouton poussoir S1 est enfoncé. Lorsque le bouton poussoir est relâché, l'incrément s'arrête. Lorsque le bouton poussoir

S2 est enfoncé, il initie un décrément de un. Lorsque le nombre est de 99 et que S1 est enfoncé, il devient 00. Lorsque le nombre est de 00 et que S1 est enfoncé, il deviendra 99.

TROSIEME E PARTIE : TIMER0:

-Saisir le code suivant sur l'éditeur du compilateur CCS ,puis compiler et démarrer la simulation sous Proteus. La LED commencera à clignoter normalement.

- Vérifier par simulation et Montrer par calcul que Timer0 déborde chaque 49.92ms (environ 50ms)
-Vérifier à l'oscilloscope que la période du signal carré sur la broche RB0 est d'environ 500ms. Montrer ça par calcul.

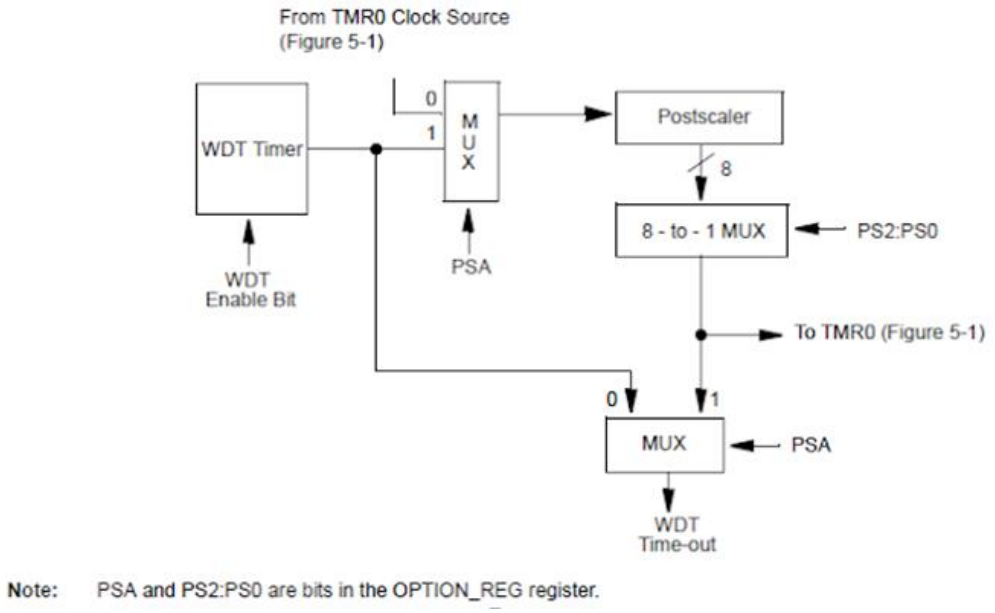



```
// PIC16F84A timer0 interrupt example
//******************************** http://ccspicc.blogspot.com/ **********************************
#include <16F84A.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay(crystal=4000000)
byte i ;
#INT_TIMER0
void timer0_isr(void)
{
  clear_interrupt(INT_TIMER0);    // Clear timer0 interrupt flag bit
  set_timer0(61);
  i++;
  if(i > 9)
  {
    i = 0;
   output_toggle(PIN_B0);
  }
}
void main()
{
  // Setup timer0 with internal clock and 256 prescaler
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
  set_timer0(61);              // Timer0 preload value
  clear_interrupt(INT_TIMER0);   // Clear timer0 interrupt flag bit
  enable_interrupts(INT_TIMER0); // Enable timer0 interrupt
  enable_interrupts(GLOBAL);     // Enable all unmasked interrupt
  output_low(PIN_B0);
  while(TRUE) ;                  // Endless loop
```

}

WATCHDOG :

Le **Watchdog Timer** est un oscillateur **RC**, qui n'a besoin d'aucun composant externe. Cela signifie que le **WDT** s'exécutera même si l'horloge de l'appareil a été arrêtée, par exemple, en exécutant une instruction **SLEEP**. En fonctionnement normal, le **WDT** génère une réinitialisation de l'appareil (**Watchdog Timer Reset**). Si l'appareil est en mode **SLEEP**, il se réveille et poursuit son fonctionnement normal (**Watchdog Timer Wake-Up**).



WDT joue un rôle très important non seulement contre les erreurs logicielles, mais aussi dans des situations environnementales agressives. Le **WDT** offre la possibilité de rétablir le contrôle de l'application par le microcontrôleur en redémarrant le système (un simple **RESET**) même si le défaut est dans l'oscillateur du microcontrôleur. L'incapacité de ce microcontrôleur à maintenir correctement le déroulement du programme peut avoir diverses causes telles que des conditions environnementales (températures extrêmes), une instabilité de puissance, une panne d'oscillateur ou des bugs de code (boucle infinie, par exemple).

Le temps du **WDT**est de 18 ms et peut être incrémenté jusqu'à quelques secondes via le pré-diviseur. Ce diviseur de fréquence est une ressource partagée entre **WDT** et **TIMER0.** Pour empêcher le système de redémarrer, vous devez ajouter l'instruction **clrwdt** à une boucle du programme afin qu'elle s'exécute périodiquement et dans une période plus courte que le temps de débordement de **WDT** .
La détection de temporisation se fait en vérifiant le bit **TO** du registre **STATUS** (pic16f).Ainsi lorsqu'il y a un débordement dans le temporisateur WDT, **TO** sera égal à 0.

Vous devez activer le **WDT** en réglant les fusibles.

TACHE : Traduire le pseudocode suivant en programme C-CCS, faire les simulations nécessaires et vérifier le bon fonctionnement du code. Combien vaut le temps de réinitialisation du pic ?.

```
Début programme
Char  k = 1
{
OPTION_REG = 0x0E; //
TRISB = 0 ;
CLRWDT; // mettre à zéro  la minuterie du chien de garde

toujours faire
   {
 CLRWDT; // mettre à zéro la minuterie du chien de garde
```

```
 tant que(k>0)
PORTB =k-1// après un certain temps, la réinitialisation se produira
Attendre  200ms
k=k+1
 Fin tant que
 PORTB = 0;
    }
Fin programme
```

NB :Pour chaque tâche que vous aurez à réaliser, vous êtes invités à enregistrer votre programme sous un nom particulier pour garder une sauvegarde de travail. Pour faciliter le débogage et assurer la maintenance et la portabilité de vos codes, il est important de commencer par rédiger un organigramme, que vous **commentiez** clairement votre programme et que vous pensiez à le rendre le plus **clair** possible .

Concernant l'évaluation de votre travail, vous devez présenter le programme correspondant à chaque **tâche** que vous aurez à programmer sur le simulateur ou sur le kit de développement.

Pour faciliter la correction de vos programme, une attention particulière est portée à la **lisibilité** (pensez donc aux commentaires !). Enfin, vous devrez également accompagner les **codes** par les **organigramme**s qui leurs correspondent.