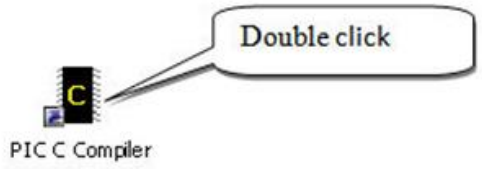
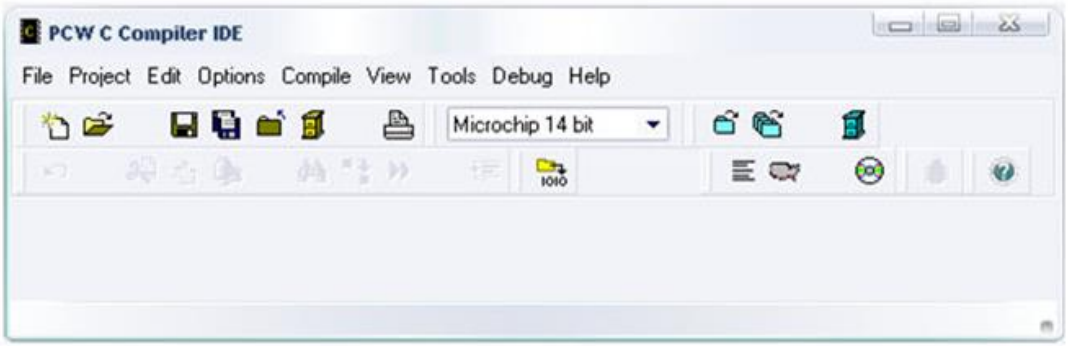
Le Compilateur CCS

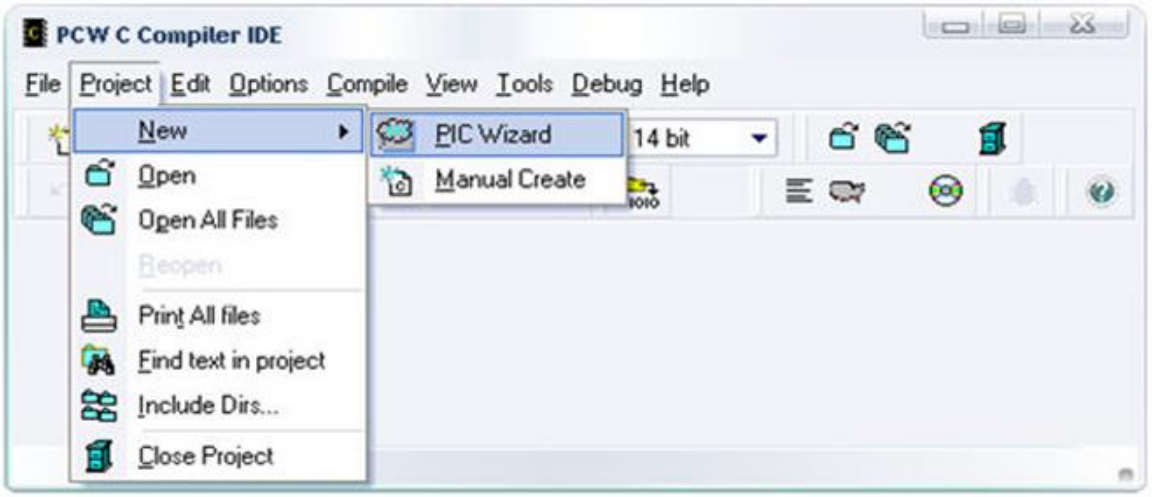
Le compilateur C de la société CCS (Custom Computer Services) est un compilateur C adapté aux microcontrôleurs PICs .Il ne respecte pas complètement la norme ANSI, mais il apport des fonctionnalités très intéressantes.



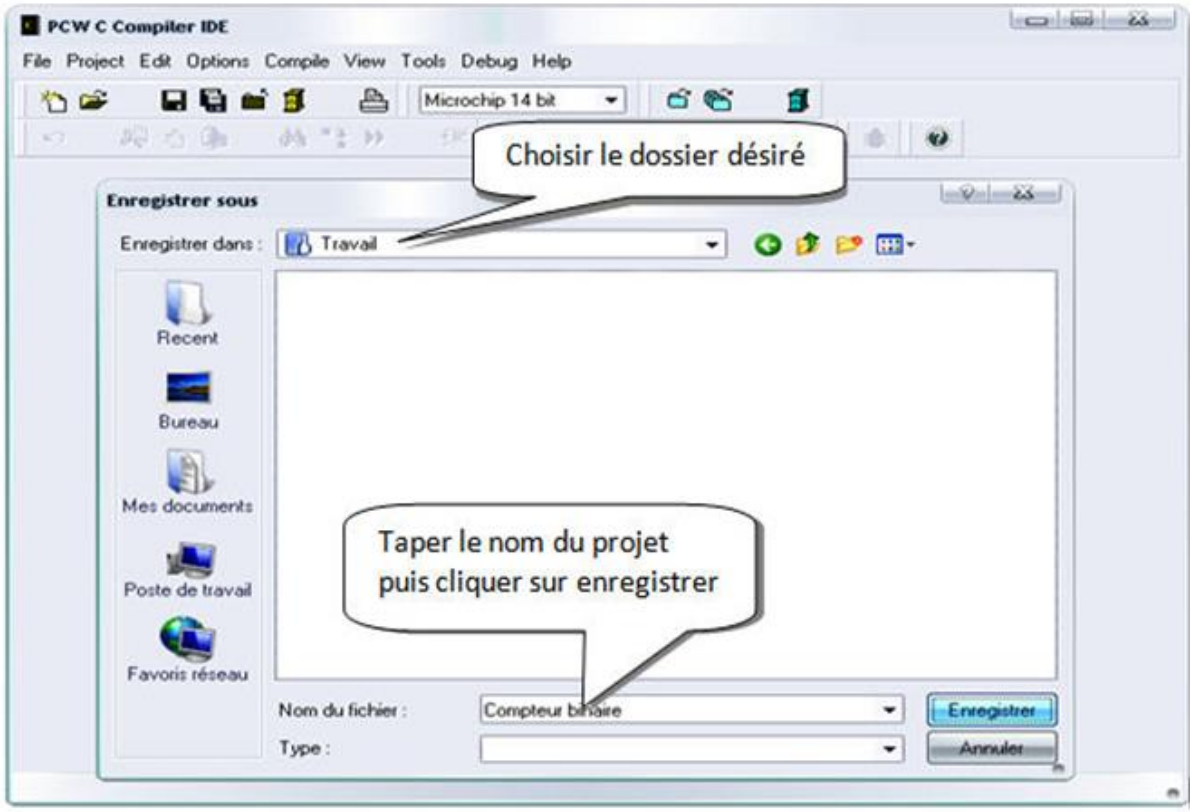
1. La démarche à suivre pour créer un projet avec le compilateur CCS :



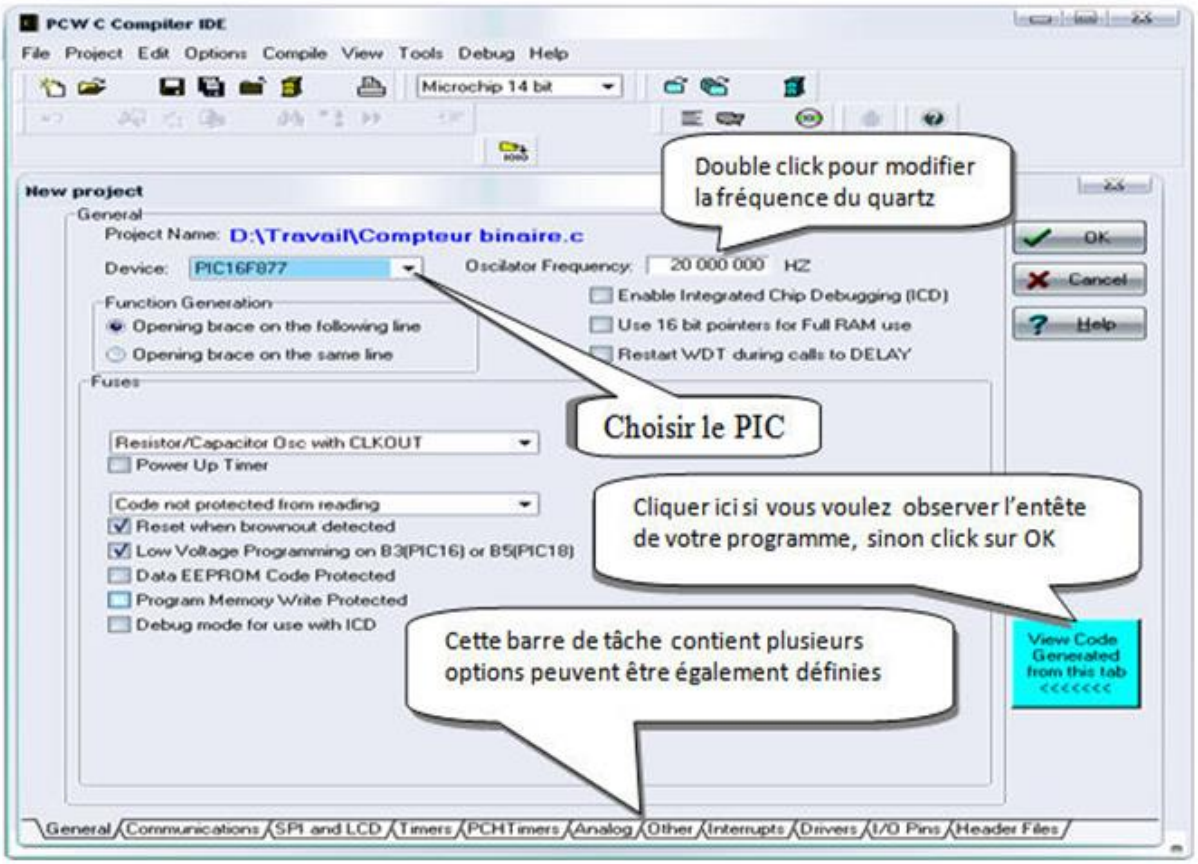
1.1. Lancement de PIC C Compiler :

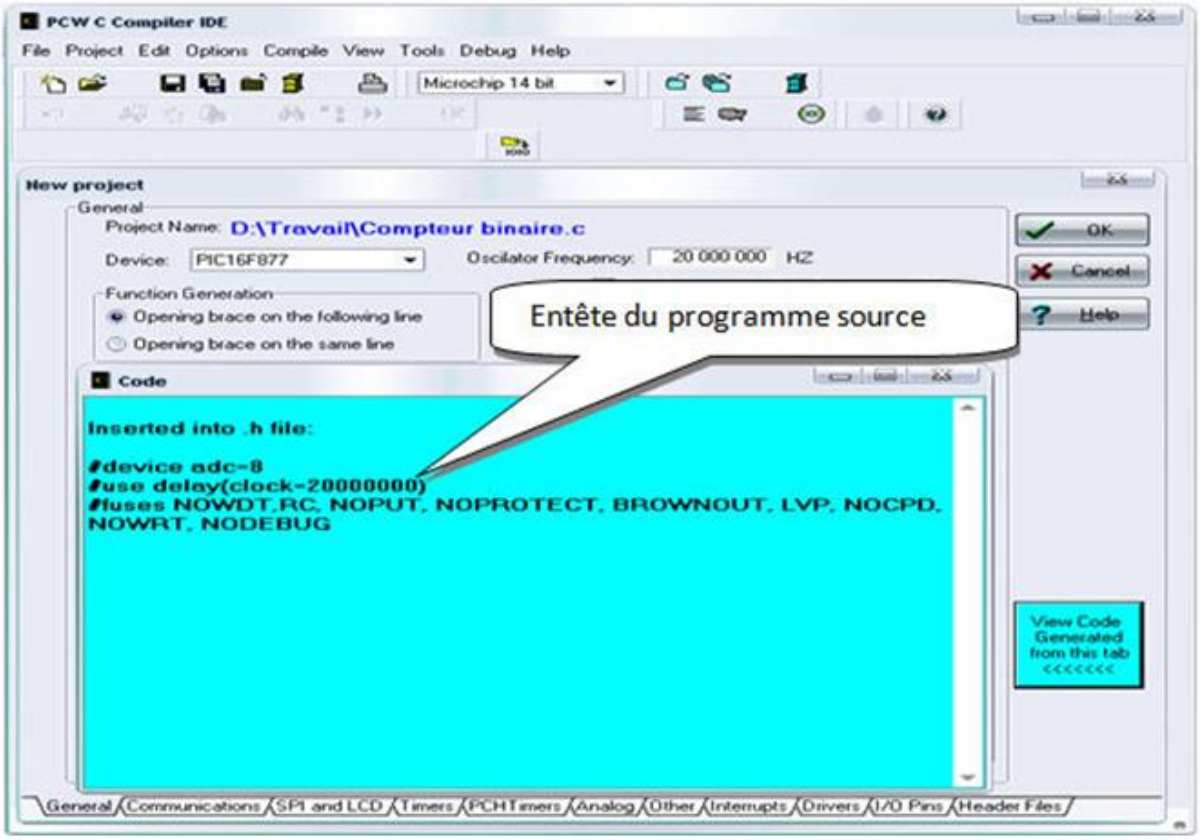


1.2. Création d'un nouveau projet : project /new /PIC Wizard

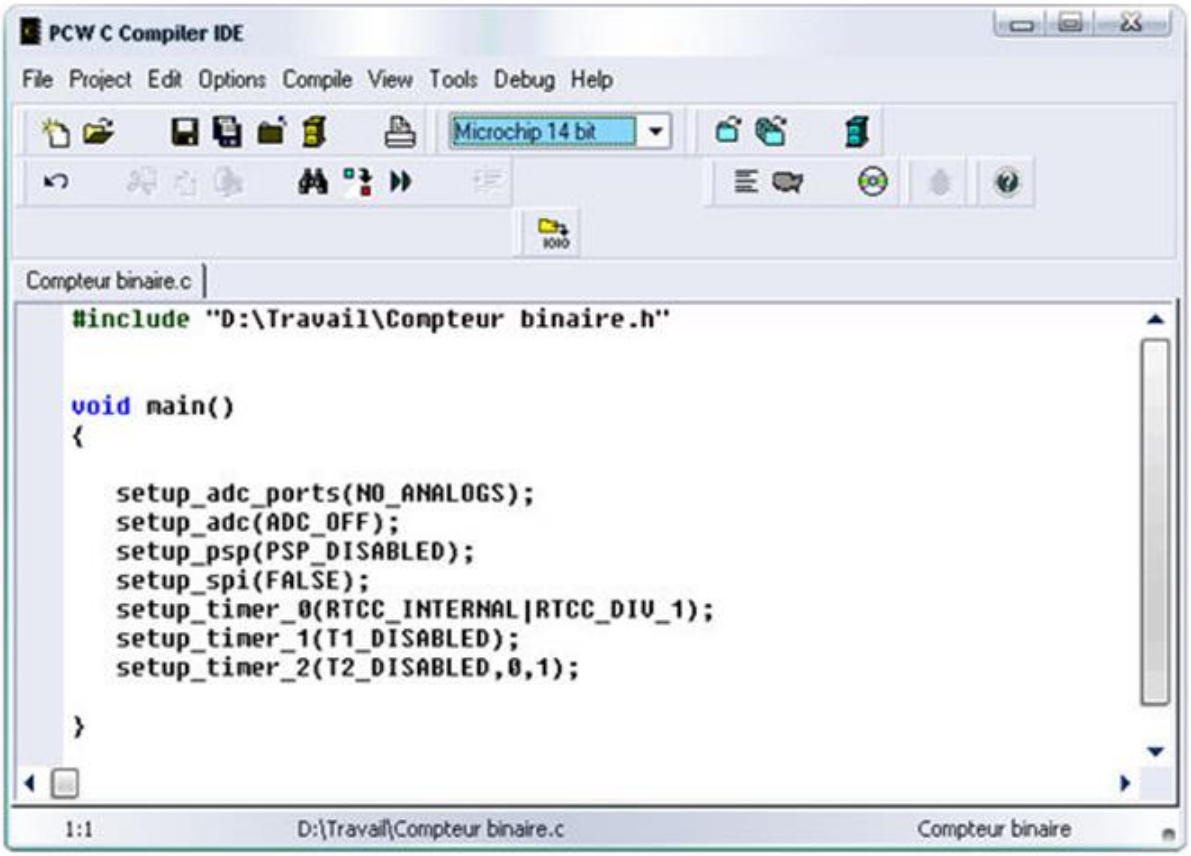


1.3. Choix du dossier de travail et du nom du projet :

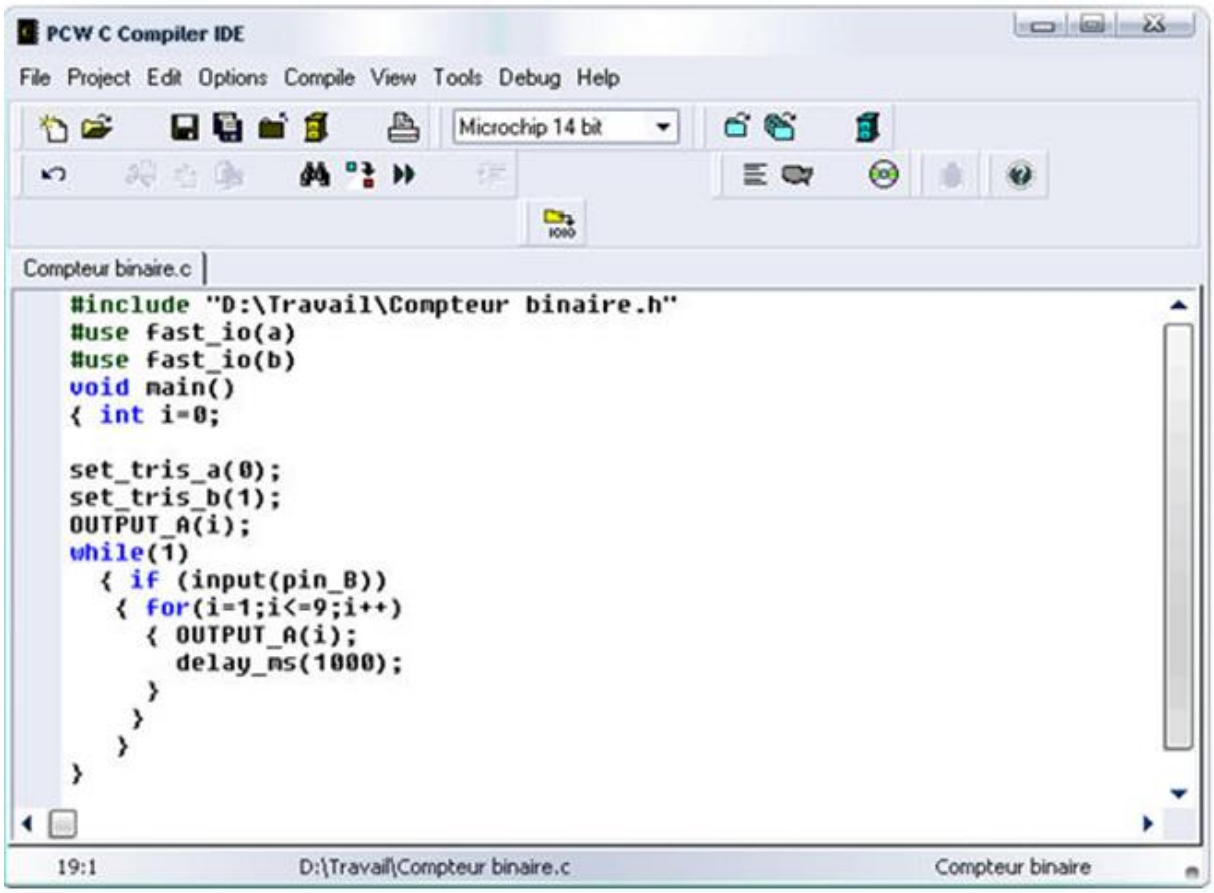




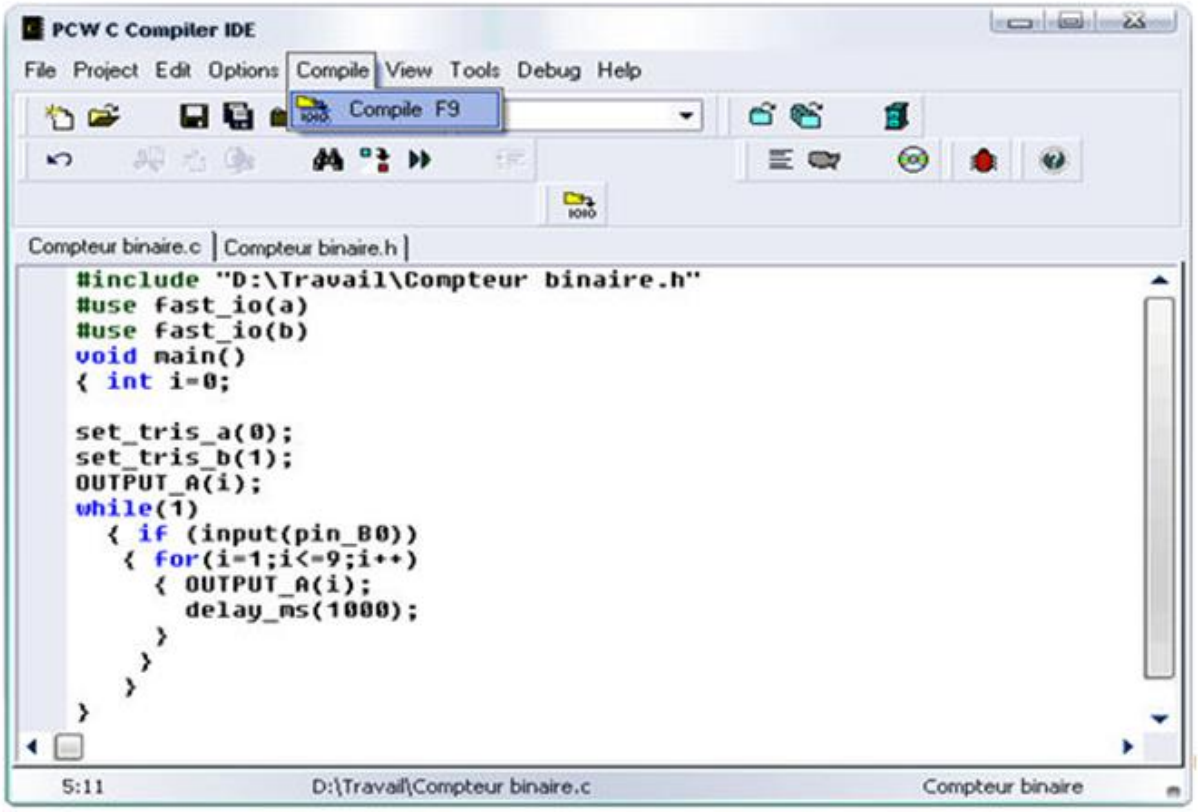
1.4. Choix des paramètres du projet :

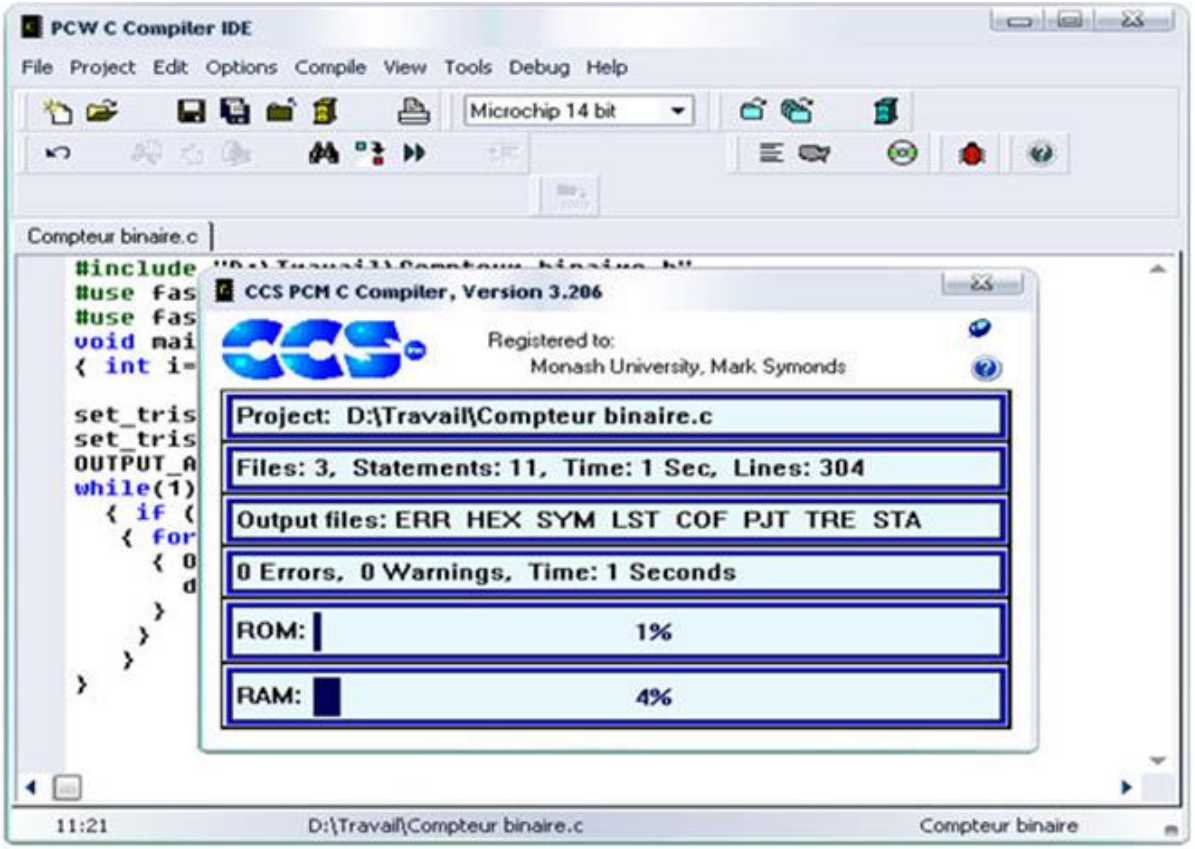


1.5. Saisie du programme :

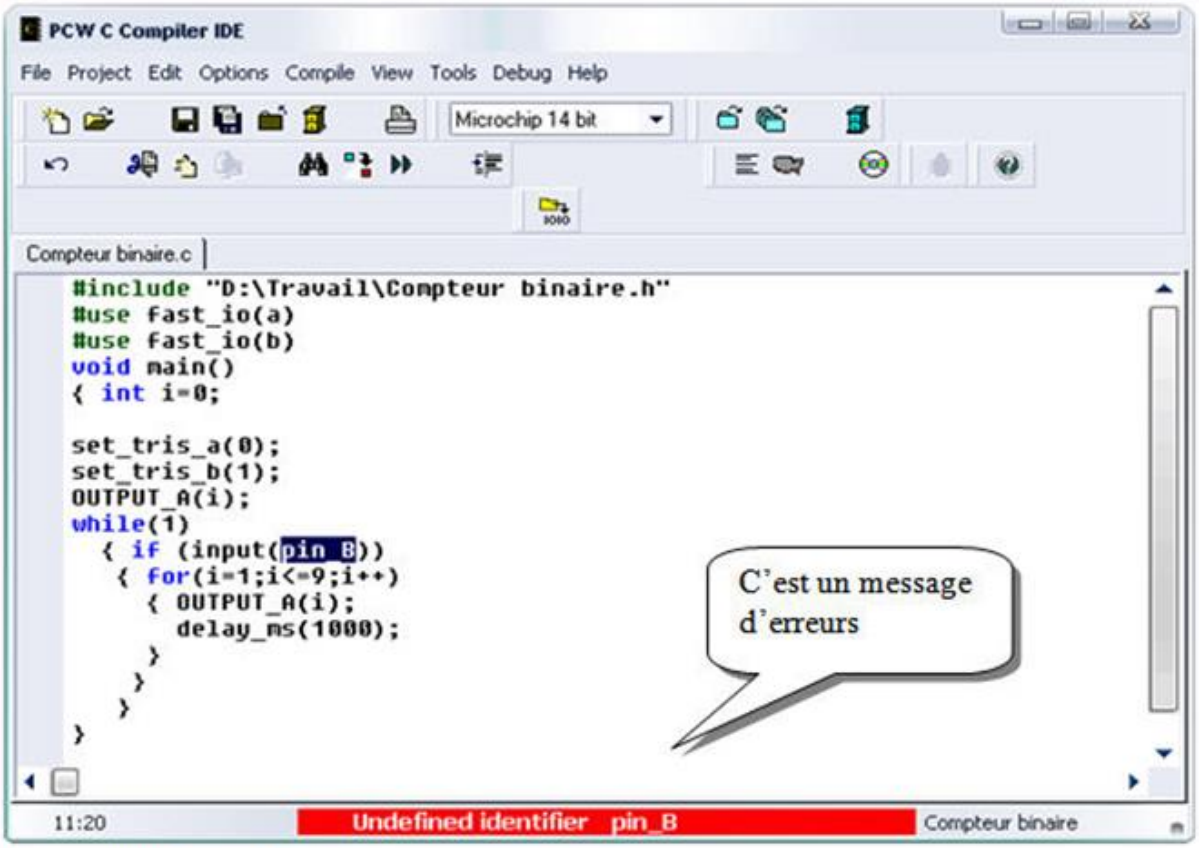


1.6. Compilation : compile compile F9

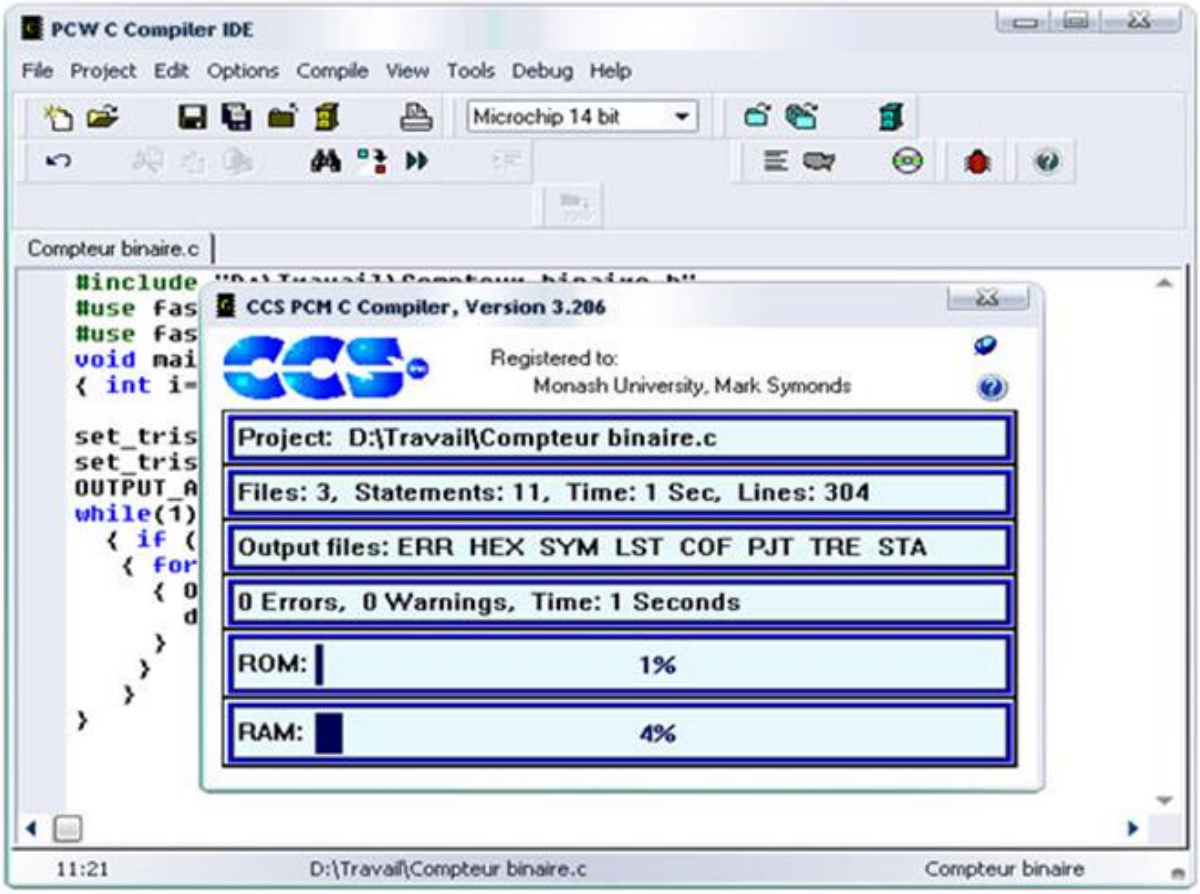




Après la compilation :

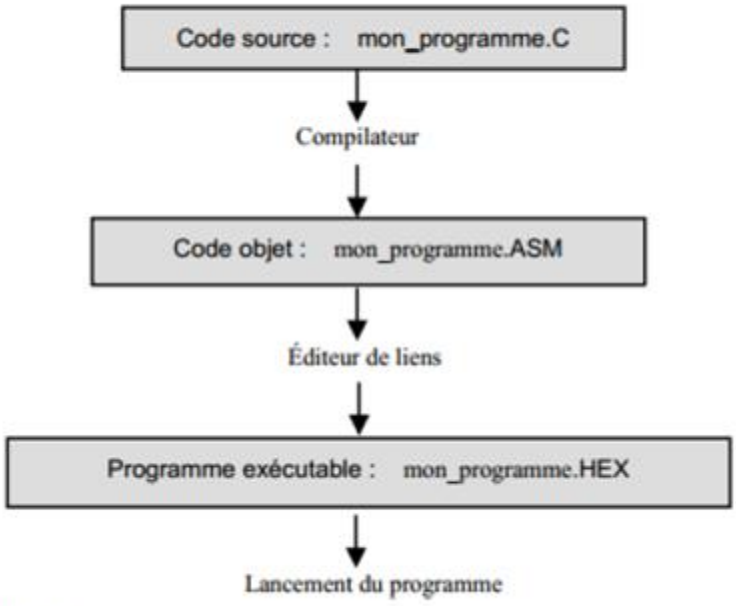


2ème compilation après la correction des erreurs :

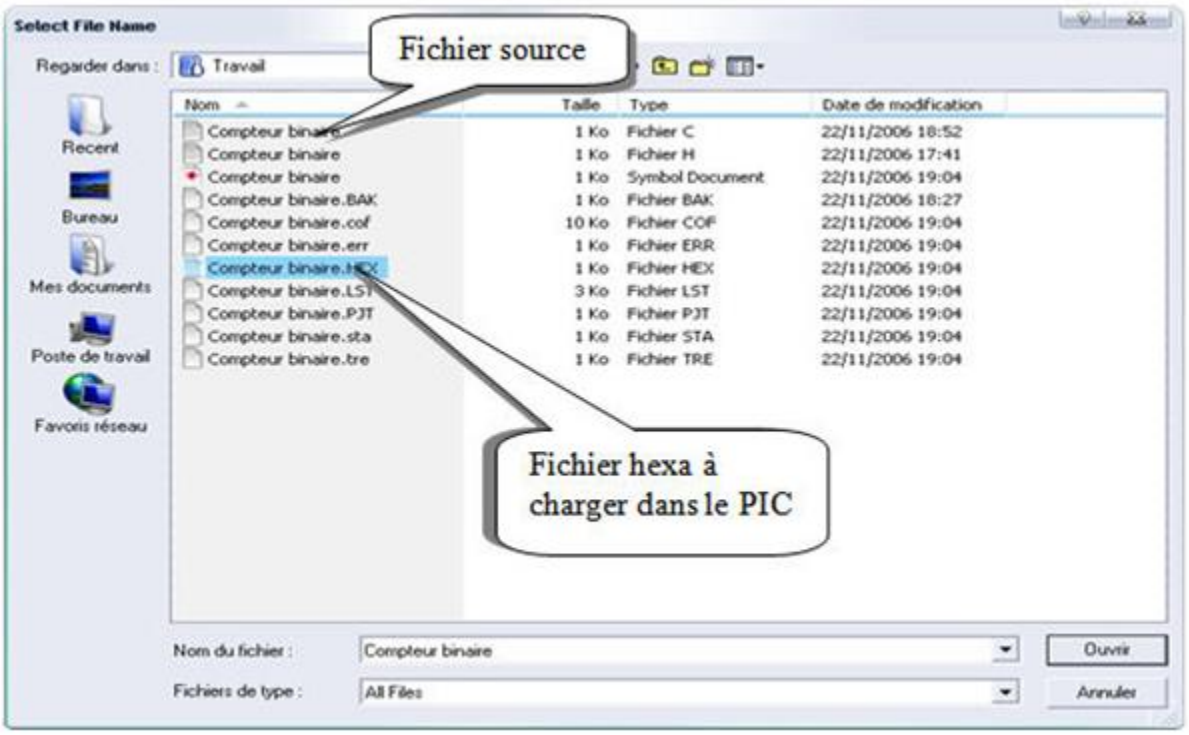


Après la compilation du fichier source comme illustré ci-dessous :

Après la compilation du fichier source comme illustré ci-dessous :



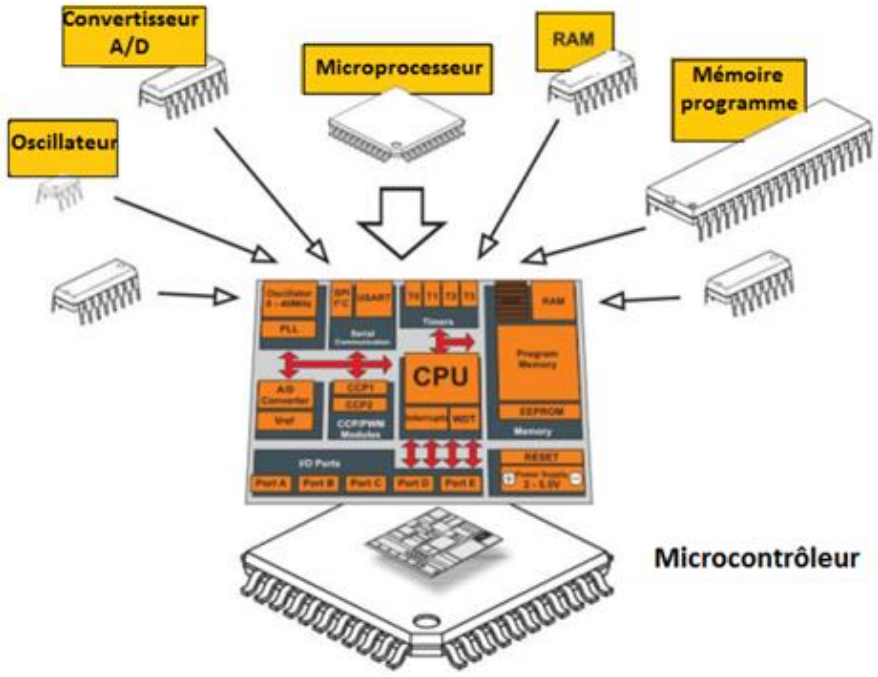
on obtient les fichiers suivants :



Fichiers (Extensions)	Description du contenu du fichier	Fichiers (Extensions)	Description du contenu du fichier
.C	Fichier source en langage C.	**.HEX**	Code objet (exécutable) téléchargé dans le µC
.H	Entête de définition des broches, Registres, Bits de Registres, Fonctions, et directives de pré-compilation.	**.TRE**	Montre l'organisation du programme sous forme d'arbre (découpages en fonction) et l'utilisation de la mémoire pour chaque fonction.
.PJT	Fichier de projet (pas obligatoire).	**.COF**	Code machine + Informations de débugage
.LST	Fichier qui montre chaque ligne du code C et son code assembleur associé généré.	**.ERR**	Erreurs éventuelles rencontrées durant la compilation.
.SYM	Indique la correspondance entre le nom des symboles (variables, bits, registres) et leur adresses hexadécimale en mémoire.		
.STA	Fichier statistique sur l'espace mémoire occupé. etc.		

Programmation en C - CCS PCWH

La programmation des microcontrôleurs PIC avec PIC C Compiler PCWH nécessite la poursuite des démarches progressives qu'il faut suivre pour élaborer un programme en C destiné pour des applications à base de PIC.



1. Les Règles de bases :

- Toutes instructions ou actions se terminent par un point virgule ;
- Une ligne de commentaires doit commencer par /* et se terminer par */ ou commence par // norme C++.
- Un bloc d'instructions commence par **{** et se termine par **}** .

2. Les variables et les constantes :

2.1. Les constantes :

Les constantes n'existent pas, c'est-à-dire qu'il n'y a pas d'allocation mémoire, mais on peut affecter à un **identificateur** (Nom : Il ne doit pas dépasser 32 caractères, sans accent)

Une valeur constante par l'instruction **#define**.

Syntaxe : < #define> <identificateur> <valeur> ;

Exemple :

#define PI 3.14;

2.1.1. Déclarations spécifiques au compilateur CCS :

- #bit id =x,y
  - Id : identifiant (Nom d'un bit)
  - X : Nom du variable ou d'une constante
  - Y : position du bit

Exemple :

#bit RW =PORTA,2

#bit BUZZER =PORTD,7

#byte id = X

- Id: identifiant
- X: valeur 8 bits

Exemple :

```
#byte  PORTA  = 5          // adresse  du port A
#byte  PORTB  = 6          // adresse  du port B
#byte  PORTC  = 7          // adresse  du port C
#byte  PORTD  = 8          // adresse  du port D
#byte  PORTE  = 9          // adresse  du port
```

2.2. Les variables :

Les variables sont définies par signé ou non signé,

Syntaxe : <Signed><type><identificateur1>,.., <identificateurn>

- **l'identificateur** : C'est le nom (il ne doit pas dépasser 32 caractères sans accent) affecté à la variable.
- **le type** : il détermine la taille de la variable et les opérations pouvant être effectuées. On peut rajouter le mot **signed** devant le type de la variable, alors les variables deviennent signées.

2.2.1Les types du compilateur CCS :

Type	Taille	valeurs
Int1	1 bit	0 OU 1
Int8	8 bits	De 0x00 à 0xFF ou 0 à 255
Int16	16 bits	De 0x0000 à 0xFFFF ou 0 à 655535
Int32	32 bits	De 0x0000000 à 0xFFFFFFFF
char	8 bits	De 0 à 255 ou une lettre A…… Z
float	32 bits	Format flottant
short		Même type que int1
int		Même type que int8
long		Même type que int16

Exemples :

Int A,B,C,D ;

Char MESSAGE[10] ;

Les types signés

Par défaut, tous ces types de données sont signés, ils peuvent être signés en rajoutant le mot clé **signed** devant le type.

Exemple :

Signed int A ; // Entier de type signé, de -128 à +127

Signed long NB; // Entier de type signé, de -32768 à +32767

2.2.2 Les base du compilateur CCS :

- Le décimal : A=10 ;
- L'octale : A=012 ;
- L'hexadécimal A=0x0A ;
- Le binaire A=0b00001010 ;

Le caractère : Exemple la lettre A code ASCII 65(Décimal) ou $41(Hexadécimal), peut s'écrire :

LETTRE = 65 Ou LETTRE = 0x41 Ou LETTRE = 'A'

3. Les opérateurs du langage C :

Lors de son exécution, un programme est amené à effectuer des opérations qui peuvent être purement arithmétiques, de comparaison, d'affectation, etc....

3.1. L'opérateur d'affectation :

Type	Symbole	Exemple
Opérateur d'affectation	=	X=10 ; Y = a + b

L'opérande de gauche prend pour valeur l'opérande de droite.

3.2. Les opérateurs arithmétiques :

Type	Symbole	Exemple
Addition	+	a = a + b ; x = 5+ a
Soustraction	-	a = a – b ; y = c - 5
Moins unaire	-	a = - b
Multiplication	*	a = a * a ; b = y * 8
Division	/	c = 9 / b ; d = a / b
Reste de la division entière	%	r = a % b

3.3. Les opérateurs de comparaison ou relationnels :

Type	Symbole	Exemple
Egalité	==	a == b ; c == 9
Différent	!=	c!= a
Supérieur	>	a > b ; 8 > a
Supérieur ou égal	>=	a >= b ; 8 >= a
Inférieur	<	a < b ; 8 < a
Inférieur ou égal	<=	a <= b ; 8 <= a

3.4. Les opérateurs logiques :

Type	Symbole
Et logique	&&
Ou logique	||
Non logique	!

3.5. Les opérateurs binaires bit à bit :

Type	Symbole	Exemple
Et binaire	&	x = y & z
Ou binaire	|	x = y | z
Ou exclusif	^	x = y ^ z
Complément à 1	~	x = b ~ z

Décalage de n bits à droite	>>	x = b >> n
Décalage de n bits à gauche	<<	x = b << n

4. Les structures répétitives.

Le langage "C" possède des instructions permettant de répéter plusieurs fois une même séquence en fonction de certaines conditions.

4.1 Structure "while" : tant que ... faire ...

Avec ce type d'instruction le nombre de répétitions n'est pas défini et dépend du résultat du test effectué sur la condition. Si cette dernière n'est jamais vérifiée, la séquence n'est pas exécutée.

```
while (int x!=0)
        {
            ...
        }
```

La structure précédente répète la suite d'instruction comprises entre crochets tant que la variable entière "x" est différente de 0.

4.2 Structure "do ... while" : faire ... tant que...

Cette structure ressemble fortement à la précédente à la seule différence que la séquence à répéter est au moins exécuter une fois même si la condition n'est jamais vérifiée.

```
do  {
            ...
          }
   while (int x!=0);
```

4.3 Structure "for" : Pour <variable> allant de <valeur initiale> à <valeur finale> faire...

Cette instruction permet de répéter, un nombre de fois déterminé, une même séquence.

```
for (i=0;i<10;i++)
          {
              ...
          }
```

La structure précédente répète 5 fois la suite d'instruction comprise entre crochets. La variable "i" prendra les valeurs successives de : 0, 1, 2, 3 et 4.

4.4 Les structures alternatives.

Ces structures permettent d'exécuter des séquences différentes en fonction de certaines conditions.

4.5 Structure "if ... Else" : Si <condition> faire ... sinon faire ...

Avec cette structure on peut réaliser deux séquences différentes en fonction du résultat du test sur une condition.

```
if (a<b) c=b-a;
          else c=a-b;
```

La structure précédente affecte la valeur "b-a" à "c" si "a" est inférieur à "b" sinon "c" est affecté par la valeur "a-b".

4.6 Structure "switch ... case".

Cette structure remplace une suite de "if ... else if ... else" et permet une de réaliser différentes séquences appropriées à la valeur de la variable testée.

```
switch (a)
   {
  case '1' : b=16;
  case '2' : b=8;
  case '3' : b=4;
  case '4' : b=2;
   }
```

Dans la structure précédente "b=16" si "a=1", "b=8" si "a=4" etc.

5. Les fonctions adaptées aux microcontrôleurs PIC :

5.1. La gestion des entrées et des sorties :

Les fonctions suivantes permettent d'agir sur les ports d'entrées et de sorties :

- **Output_low ( ) ;**
- **Output_high ( ) ;**
- **Output_bit (pin_xx, 0 ou 1) ;**
- **Input (pin_xx) ;**
- **Output _x (valeur) ; // X : nom de port A…E**
- **Input_x ( ) ;**
- **Set_tris_x (valeur) ; // Valeur : définie la configuration de port (1 entrée et 0 sotie)**

Exemples :

Lecture de la broche Rb4 :

```
X = input (pin_B4) ;
```

Mise à 1 de RA5 :

```
output (pin_A4 ,1) ;
```

ou

```
output_high (pin_A4) ;
```

Configuration de port c, 4 broches en entrées et 4 broches en sorties :

```
Set_tris_c (0b11110000) ;
```

5.2. La gestion des temporisations :

Le compilateur intègre des fonctions très pratique pour gérer les délais :

- **delay_cycles (valeur) ; // temporisation en NB de cycles**
- **delay_us (valeur) ; // temporisation en µs**
- **delay_ms (valeur) ; // temporisation en ms**

Pour pouvoir utiliser ces fonctions, il faut indiquer la fréquence du Quartz de votre application, cette ligne doit être définie au début du programme.

```
#use delay (clock = fréquence_de_quartz) ;
```

Exemples :

```
#use delay (clock=4000000);      // quartz de 4 MHz
#use delay (clock = 20000000);   //quartz en 20 MHz
```

5.3. La gestion de la liaison série :

Toutes les fonctions d'entrée et de sortie peuvent être redirigées sur le port série du microcontrôleur, il suffit d'ajouter la ligne ci-dessous pour configurer le port série :

#use rs232 (BAUD=9600, xmit=PIN_C6, rcv=PIN_C7) //Cette ligne configure la liaison série de PIC avec une vitesse de 9600 bauds.

Les fonctions suivantes utiliseront le port série comme moyenne de communication :

- Printf() ;
- Putc() ;
- getc() ;
- Kbhit() ;

Traduction d'un algorithme en langage C :

1. Constantes et variables :

1.1. Déclaration de Constantes :

Algorithmique : Constante Nomconstante : [Type] = valeur

Syntaxe C : #define ID de la constante valeur

Exemple :

Affectation d'E/S de type bit : commande d'un moteur

```
#bit LIMIT_=PORTC,1;
#bit CLOCK_=PORTD,3;
#bit F_H   =PORTC,5;
#bit ENABLE =PORTC,0;
```

1.2. Déclaration de variables:

Algorithmique : **variable** Nomvariable : [Type]

Syntaxe C : #define ID de la variable :

Exemples :

```
float VALEUR_CAN ;        // variable de type réel
int ETAT_RE[4] ;          // tableau de 4 éléments entier
char  MESSAGE_N1[10] ;    // chaîne de 9 caractères
```

2. Procédures et fonctions :

La traduction d'une fonction ou procédure ou encore appelé sous-programme s'effectue ainsi :

2.1. Syntaxe :

```
//Nom de la fonction :
//Description du rôle de la fonction :
//Paramètres d'entrée : Noms et types des paramètres d'entrée
//Paramètre de sortie : Nom et type du paramètre de sortie
Type de la variable de retour   nom de fonction (types nom des paramètres)
```

Type de la variable de retour nom de fonction (types nom des paramètres)

{

Instruction 1 ;

.

.

Instruction n ;

Return (valeur) ; // Valeur à renvoyer

}

- A partir de ce syntaxe, on peut avoir plusieurs cas possibles comme :

2.2. Une fonction sans paramètres d'entrée et de sortie :

//Nom de la fonction :

//Description du rôle de la fonction :

//Paramètres d'entrée : Rien

//Paramètre de sortie : Rien

Void nom de fonction (Void)

{

Instruction 1 ;

.

.

.

Instruction n ;}

2.3. Une fonction avec des paramètres d'entrée et sans paramètre de sortie :

//Nom de la fonction :

//Description du rôle de la fonction :

//Paramètres d'entrée : Noms et types des paramètres d'entrée

//Paramètre de sortie : Rien

Void nom de fonction (types nom des paramètres)

{

Instruction 1 ;

.

.

.

Instruction n ;}

2.4. Une fonction avec des paramètres d'entrée et un paramètre de sortie :

//Nom de la fonction :

//Description du rôle de la fonction :

//Paramètres d'entrée : Noms et types des paramètres d'entrée

//Paramètre de sortie : Nom et type du paramètre de sortie

Type de la variable de retour nom de fonction (types nom des paramètres)

{

Instruction 1 ;

.

Instruction n ;

Return (valeur) ; // Valeur à renvoyer }

2.5Une fonction d'interruption :

L'exécution d'une fonction d'interruption répond a un évènement qui peut être interne (périphérique : CAN, TIMER, EEPROM, USART, I2C) ou externe (RB0,PORTB) du microcontrôleur. L'appel d'une fonction d'interruption ne dépend pas de programme principal, mais elle l'interrompe pendant son exécution.

Une fonction d'interruption n'a pas de paramètre d'entrée et de sortie. Le compilateur CCS utilise une directive spéciale **INIT_XXXX** (**XXXX** nom de l'interruption) pour les différencier avec les autres fonctions logicielles.

Syntaxe :

//Nom de la fonction :

//Description du rôle de la fonction :

#INIT_XXXX //Nom de l'interruption

Void nom de fonction (Void)

{

Instruction 1 ;

.

Instruction n ;

}

Organisation d'un programme en langage C :

```
////////////////////////////////////////////////////////////////////////////////
//rôle du programme :                                                         //
// Auteur :                                                                   //
//Lieu :                                                                      //
//Version :                                                                   //
////////////////////////////////////////////////////////////////////////////////

//fichier de déclaration des registres internes du microcontrôleur 16F84A.H
#include <16F84A.H>

//Déclaration des adresses des ports E/S
#byte PORTA = 5                    //adresse du port A
#byte PORTB = 6                    // adresse du port B
//Déclaration des constantes
#define NB_MAX 100
//Affectation des entrées et sorties de type bit
```

```
#bit  BUZZER = PORTD7          //par exemple : Command d'un buzzer
//Fréquence du quartz
#use delay (clock=20000000)
//Configuration de la liaison série de PIC avec une vitesse de9600 bauds
#use rs232 (BAUD=9600, xmit=PIN C6, rcv=PIN C7)
//Déclaration du prototype de toutes les fonctions logicielles
//Nom de la fonction suivi d'un point virgule par exemple :
Void INIT UC(Void) ;
//Déclaration des variables globales Par exemple :
Long DEBIT ;
Long VOULUME ;
//Programme principale
Main()
{
       Instruction 1 ;
       .
       Instruction n ;
}
//Déclaration des fonctions logicielles Par exemple :
//Nom de la fonction : DECALAGE DROITE
//Description du rôle de la fonction : décalage à droite de NB bits
//Paramètres d'entrée : entier VAL, entier NB
//Paramètre de sortie : entier RESULTAT
int  DECALAGE DROITE (int VAL ,int NB)
{
      Int RESULTAT;
      RESULTAT = VAL >> NB;
    Return (RESULTAT);}
//Appel de la fonction: Nom de la fonction (nom des paramètres) ;
//Exemple :
A = 16 ;
B = DECALAGE DROITE (A, 2) ;     //la valeur de A sera affectée à VAL
                                 //la valeur 2  sera affectée à NB
                                 //le résultat de la fonction sera affectée à B
```

Quelques directives et fonctions du pic C compiler

1- Directives

#use delay

Syntaxe	:	#use delay(***clock=fréquence***)
Rôle	:	Renseigne le compilateur sur la fréquence du quartz utilisé.
Exemple	:	#use delay(clock=4000000)

#fuses

Syntaxe	:	#fuses ***options***
Rôle	:	Permet de définir le mot de configuration. Les options sont :

- LP, XT, HS, RC
- WDT, NOWDT
- PUT, NOPUT

- PROTECT, NOPROTECT.

Exemple : #fuses XT,NOWDT,NOPUT,NOPROTECT

-

#int_xxxx

Syntaxe : #int_ext : interruption externe.
#int_ RB : changement d'état de RB4 à RB7.
#int_TIMER0 : débordement du timer0.
#int_EEPROM : fin d'écriture dans l'EEPROM.

Rôle : Spécifie la source de l'interruption.

2- Fonctions

BIT_CLEAR()

Syntaxe : BIT_CLEAR(***var, bit***)

Rôle : Mettre à 0 le bit « bit » de la variable « var ».

Exemple : a=0x1F
BIT_CLEAR(a, 3)
a devient 17 hexa.

BIT_SET()

Syntaxe : BIT_SET(***var, bit***)

Rôle : Mettre à 0 le bit « bit » de la variable « var ».

Exemple : a=0x1F
BIT_SET(a, 6)
a devient 3F hexa.

BIT_TEST()

Syntaxe : BIT_TEST(***var, bit***)

Rôle : Teste l'état du bit « bit » de la variable « var ».

Exemple : a=0x1F
BIT_TEST(a, 2)
Le résultat est 1 car le bit 2 de la variable « a » est à 1.

-

DELAY_MS()

Syntaxe : DELAY_MS(***x***)

Rôle : Temporisation se « x » ms.

Exemple : DELAY_MS(2000)
Temporisation de 2 secondes.

INPUT()

Syntaxe : etat=INPUT(***pin***)

Rôle : Permet de lire l'état d'une broche d'un port préalablement configurée en entrée.

Exemple : a=input(PIN_B0)
la variable « a » reçoit l'état de la broche 0 du port B.

INPUT_x()

Syntaxe : etat=INPUT_x()

Rôle : Permet de lire l'état d'un port (x sur 8 bits) préalablement configuré en entrée.

Exemple : c=input_A()

la variable « c » reçoit l'état du port A (1 octet).

-

OUTPUT_x()

Syntaxe : OUTPUT_x(***valeur***)

Rôle : Permet de sortir l'octet «valeur» sur le port x, préalablement configuré en sortie.

Exemple : OUTut_B(0x1F)

La valeur 1FH est envoyée sur le port B.

-

OUTPUT_BIT()

Syntaxe : OUTPUT_BIT(***pin***, ***etat***)

Rôle : Permet de mettre la pin (***pin***) à l'état logique (***etat***).

Exemple : OUTut_BIT(PIN_A3,1)

Mettre à 1 la broche 3 du port A.

OUTPUT_HIGH()

Syntaxe : OUTPUT_HIGH(***pin***)

Rôle : Permet de mettre à 1 la pin (***pin***).

Exemple : OUTut_HIGH(PIN_A3)

Mettre à 1 la broche 3 du port A.

OUTPUT_LOW()

Syntaxe : OUTPUT_LOW(***pin***)

Rôle : Permet de mettre à 0 la pin (***pin***).

Exemple : OUTut_LOW(PIN_A3)

Mettre à 0 la broche 3 du port A.

-

ROTATE_LEFT()

Syntaxe : ROTATE_LEFT(***adresse***, ***n***)

Rôle : Rotation à gauche de « n » positions de l'octet ayant pour adresse « adresse ».

Exemple : a=0x86

ROTATE_LEFT(&a, 1)

a devient 0d hexa.

ROTATE_RIGHT()

Syntaxe : ROTATE_RIGHT(***adresse***, ***n***)

Rôle : Rotation à droite de « n » positions de l'octet ayant pour adresse « adresse ».

Exemple : a=0x86

ROTATE_ RIGHT (&a, 1)

a devient 43 hexa.

SET_TRIS_x()

Syntaxe : SET_TRIS_x(***valeur***)

Rôle : Configure la direction du port « x ».

Exemple : SET_TRIS_B(0x0F)

Les pins B7, B6, B5, B4 sont configurées en sortie (0).

Les pins B3, B2, B1, B0 sont configurées en entrée (1).

-

SHIFT_LEFT()

Syntaxe	:	SHIFT_LEFT(*adresse, n, bit*)
Rôle	:	Décalage à gauche de « n » positions de l'octet d'adresse « adresse » ; bit est le bit introduit 0 ou 1.
Exemple	:	SHIFT_LEFT(&b, 2, 0) Décalage à gauche de 2 positions de la variable « b ».
Exemple	:	SHIFT_LEFT(&b, 2, 0) Décalage à gauche de 2 positions de la variable « b ».

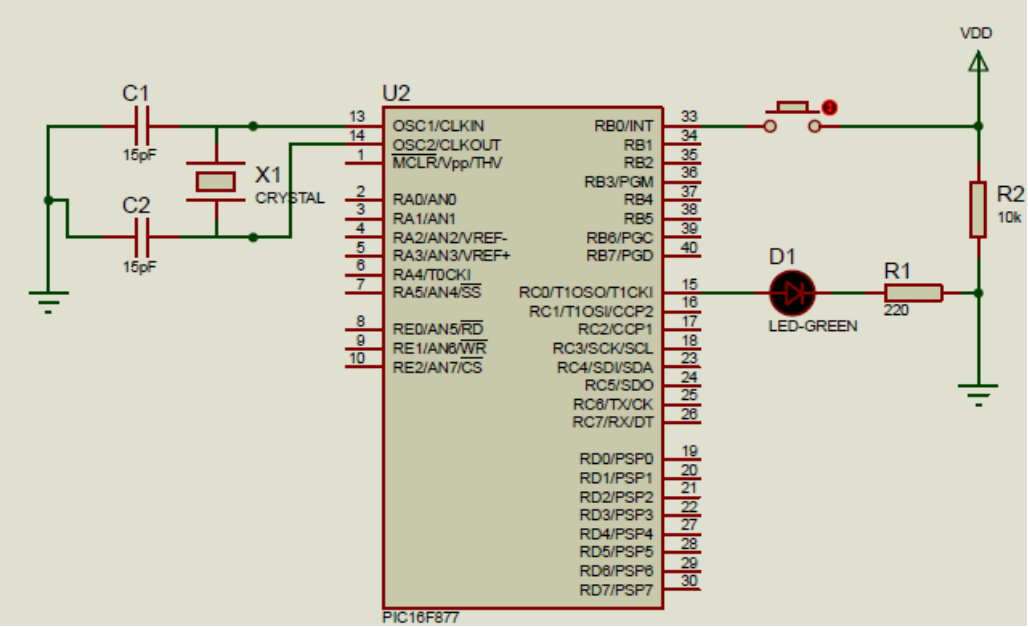
-

SHIFT_RIGHT()

Syntaxe	:	SHIFT_RIGHT(*adresse, n, bit*)
Rôle	:	Décalage à droite de « n » positions de l'octet d'adresse « adresse » ; bit est le bit introduit 0 ou 1.
Exemple	:	SHIFT_ RIGHT (&b, 2, 0) Décalage à droite de 2 positions de la variable « b ».

EXEMPLE1 : LES INTERRUPTIONS AVEC PCW CCS

l'interruption RB0

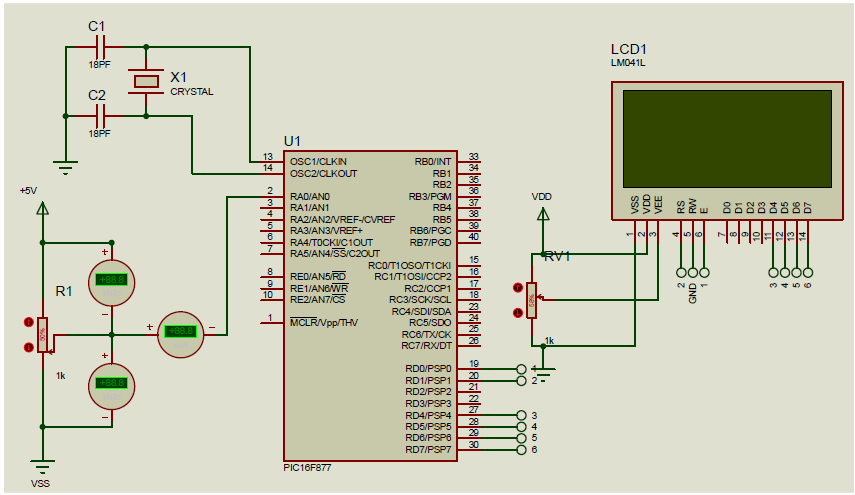



```
#include  "16F877.H"
#use delay(clock=20000000)
#define LED  PIN_C0          // Led temoin
void cligne(int x);          // prototype de la fonction cligne()
//-------------------------------------------------------//
// Sous programme de traitement de l'interruption externe
//-------------------------------------------------------//
#int_ext        // Cette directive indique que la fonction suivante est la tache de l'interruption
rb_ext( )
{
    cligne(3);
  }
```

```
//-------------------- Programme principal ----------------------
void main(void)
{
  ext_int_edge(H_TO_L);          // Front descendant
  enable_interrupts(INT_EXT);    //Valider l'interruption sur RB0
  enable_interrupts(GLOBAL);     // Valider les interruptions
  set-tris_c(0x00);              // PORTC en entrée

  while(1);
}
  //------------------------------------------------------------------------------ //
  // void cligne(int x)   // Clignoter la led verte x fois à intervalle de 1 s.
  //------------------------------------------------------------------------------ //
  void cligne(int x)
  {
  int i;
    for (i=0; i<x; ++i)
      {
        output_low(led);
        delay_ms(1000);
        output_high(led);
        delay_ms(1000);
      }
    }
```

EXEMPLE2 : LA CONVERSION ANALOGIQUE /NUMERIQUE AVEC PCW CCS



```
#include <16F877.h>
#uses delay (clock=4000000)
#include <LCD.C>          // appel du fichier contenant le driver du LCD
#device ADC=8             // resolution à 8 bits

void main()
{
  float tension,quantum;  //
  setup_adc_ports(RA0_ANALOG);          //RA0 configurée en entrée analogique
  setup_adc(ADC_CLOCK_INTERNAL);  // Utilisation de  l'horloge interne 4Mhz
  set_adc_channel(0);                   // lit sur RA0
  delay_ms(100);
  lcd_init();                           // initialisation du LCD
```

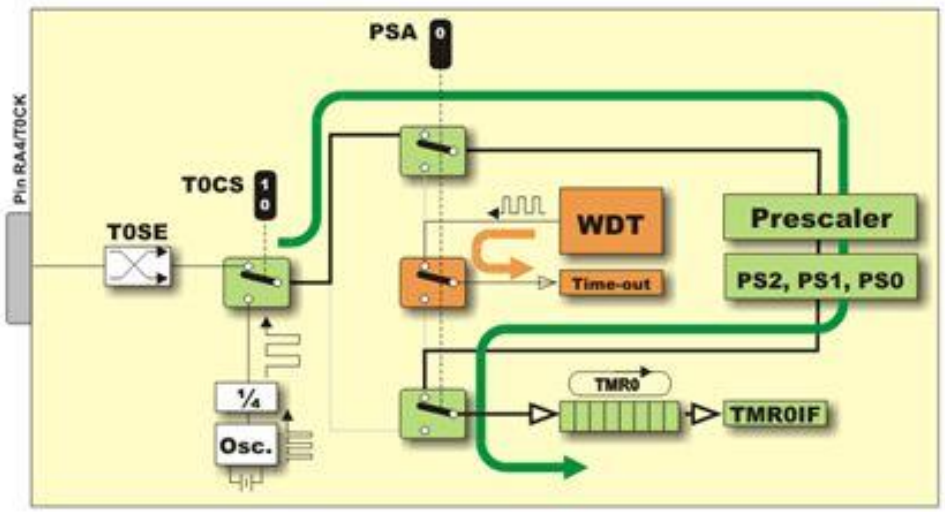
```
lcd_putc("\f*** CAN ***\n");
delay_ms(1000);
lcd_putc('\f');
quantum = 5.0/255.0;                // calcul du pas de  quantification
```

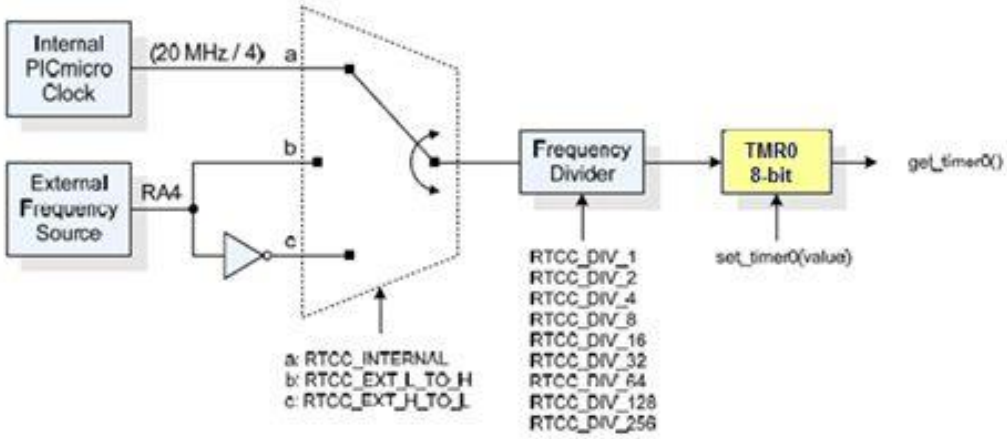
```
   while (true)
   {
   tension = read_adc()*quantum;          // lecture  de la tension convertie
   printf(lcd_putc,"\f Tension: %2.2f v \n", tension );
   printf("\f Tension = %2.2f v\n", tension );
   delay_ms(1000);
    }
}
```

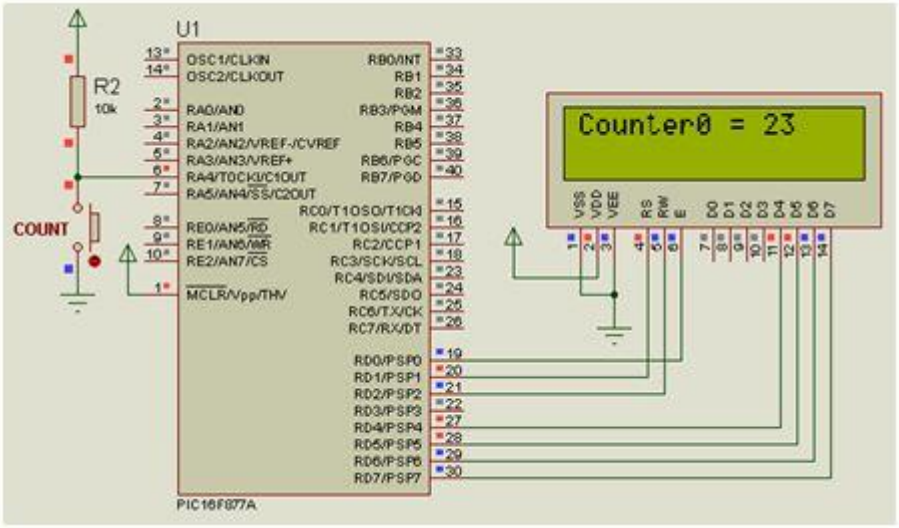
EXEMPLE3 : Timer0 avec le compilateur CCS C :

utilisation comme compteur :

La figure suivante montre le schéma de fonctionnement du Timer0 :








```
#include <16F877A.h>

#use delay(clock=20000000)

#include<lcd.c>

void main(){

int count;

//set_tris_b(0x01); /* Set I/O pins of Port B */

lcd_init();
```

```
/* Set up 8-bit RTCC */
setup_timer_0(RTCC_EXT_L_TO_H|RTCC_DIV_1);
set_timer0(0); /* TMR0 = 0 */
  while(TRUE)
  {     count = get_timer0(); /* Read counter0 */
     printf(lcd_putc ,"\fCounter0 = %u",count); /* ... send it out! */
     delay_ms(200);         /* Wait 200 ms to debounce */
}
```

QUELQUES FONCTION CCS

E/S RS232	ASSERT( )	GETCH( )	PUTC( )	
	FGETC( )	GETCHAR( )	PUTCHAR( )	
	FGETS( )	GETS( )	PUTS( )	
	FPRINTF( )	KBHIT( )	SET_UART_SPEED( )	
	FPUTC( )	PERROR( )	SETUP_UART( )	
	FPUTS( )	PRINTF( )		
E/S BUS	SETUP_SPI( )	SPI_DATA_IS_IN( )		SPI_WRITE( )
SPI 2-HILOS	SPI_XFER( )	SPI_READ( )		
E/S DISCRETAS	GET_TRISx( )	INPUT_K( )	OUTPUT_FLOAT( )	SET_TRIS_B( )
	INPUT( )	INPUT_STATE( )	OUTPUT_G( )	SET_TRIS_C( )
	INPUT_A( )	INPUT_x( )	OUTPUT_H( )	SET_TRIS_D( )
	INPUT_B( )	OUTPUT_A( )	OUTPUT_HIGH( )	SET_TRIS_E( )
	INPUT_C( )	OUTPUT_B( )	OUTPUT_J( )	SET_TRIS_F( )
	INPUT_D( )	OUTPUT_BIT( )	OUTPUT_K( )	SET_TRIS_G( )
	INPUT_E( )	OUTPUT_C( )	OUTPUT_LOW( )	SET_TRIS_H( )
	INPUT_F( )	OUTPUT_D( )	OUTPUT_TOGGLE( )	SET_TRIS_J( )
	INPUT_G( )	OUTPUT_DRIVE( )	PORT_A_PULLUPS( )	SET_TRIS_K( )
	INPUT_H( )	OUTPUT_E( )	PORT_B_PULLUPS( )	
	INPUT_J( )	OUTPUT_F( )	SET_TRIS_A( )	
E/S PUERTO	PSP_INPUT_FULL( )		PSP_OVERFLOW( )	
PARALELO ESCLAVO	PSP_OUTPUT_FULL( )		SETUP_PSP( )	
E/S BUS I2C	I2C_WRITE( )	I2C_SlaveAddr( )	I2C_ISR_STATE( )	
	I2C_POLL( )	I2C_START( )		
	I2C_READ( )	I2C_STOP( )		
CONTROL PROCESOS	CLEAR_INTERRUPT( )	GOTO_ADDRESS( )	RESET_CPU( )	
	DISABLE_INTERRUPTS( )	INTERRUPT_ACTIVE( )	RESTART_CAUSE( )	

CONTROL PROCESOS	ENABLE_INTERRUPTS( )	JUMP_TO_ISR	SETUP_OSCILLATOR( )	
	EXT_INT_EDGE( )	LABEL_ADDRESS( )	SLEEP( )	
	GETENV( )	READ_BANK( )	WRITE_BANK( )	
MANEJO BIT-BYTE	BIT_CLEAR( )	MAKE8( )	_MUL( )	SHIFT_LEFT( )
	BIT_SET( )	MAKE16( )	ROTATE_LEFT( )	SHIFT_RIGHT( )
	BIT_TEST( )	MAKE32( )	ROTATE_RIGHT( )	SWAP( )
TENSIÓN DE REFERENCIA	SETUP_VREF( )	SETUP_LOW_VOLT_DETECT( )		
A/D CONVERSIÓN	SET_ADC_CHANNEL( )		SETUP_ADC_PORTS( )	
	SETUP_ADC( )	READ_ADC( )		
TIMERS	GET_TIMER0( )	SET_RTCC( )	SETUP_TIMER_0 ( )	
	GET_TIMER1( )	SET_TIMER0( )	SETUP_TIMER_1 ( )	
	GET_TIMER2( )	SET_TIMER1( )	SETUP_TIMER_2 ( )	
	GET_TIMER3( )	SET_TIMER2( )	SETUP_TIMER_3 ( )	
	RESTART_WDT( )		SETUP_COUNTERS( )	
RETARDOS	DELAY_CYCLES( )		DELAY_US( )	DELAY_MS( )