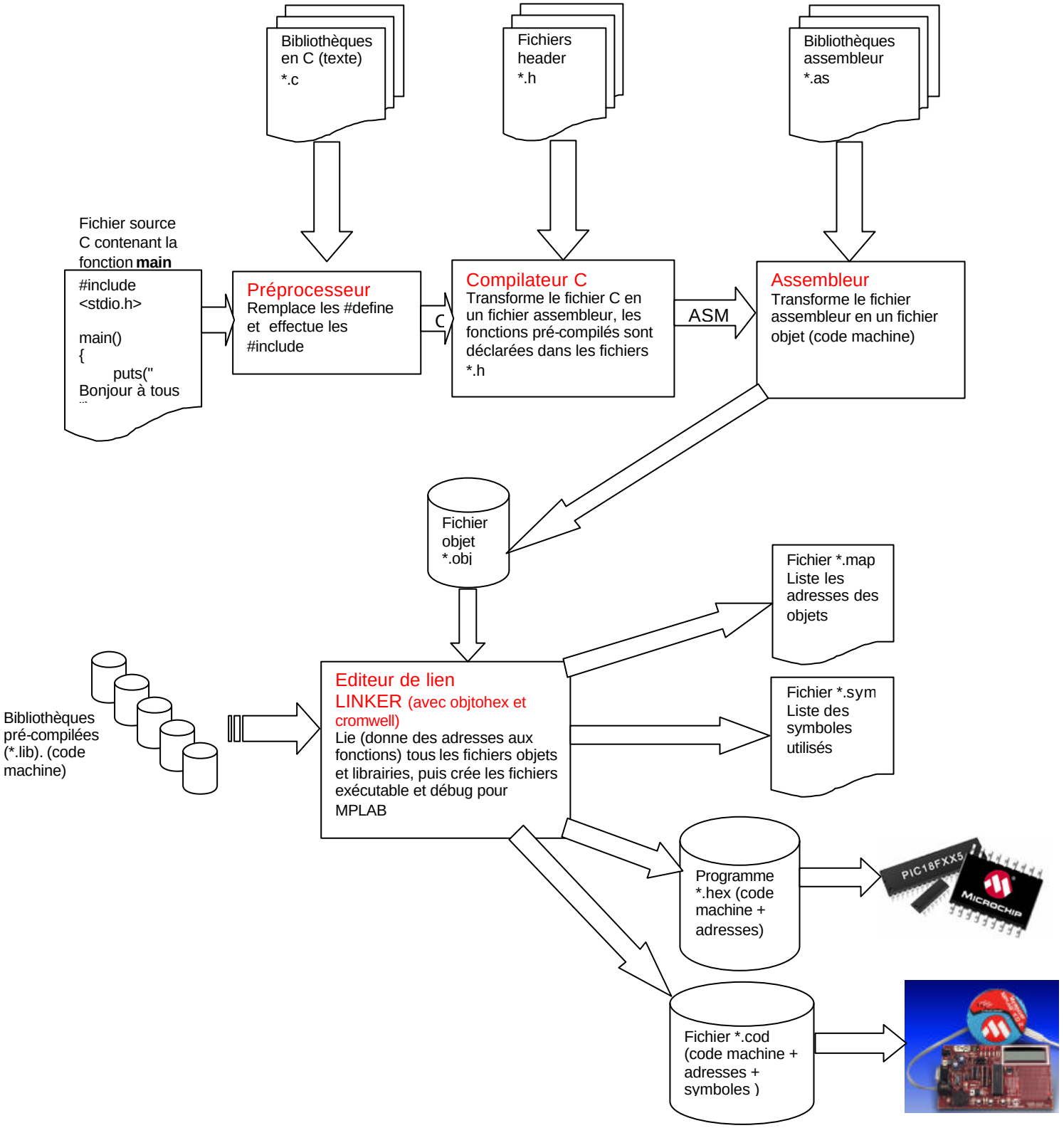
La programmation des PIC18 en langage CANSI

Christian Dupaty
Professeur d'électronique
Lycée Fourcade
13120 Gardanne
Académie d'Aix-Marseille
christian.dupaty@ac-aix-marseille.fr

Structure du compilateur C HI-TECH pour PIC18



Installation du compilateur dans l'environnement MPLAB

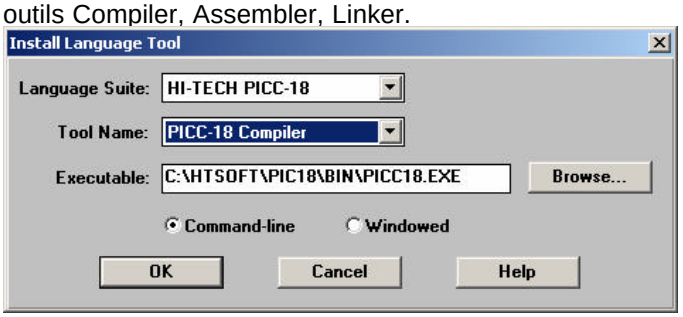
Installer le compilateur HI-TECH C18 sur le disque dur.

Pour le réinstaller :

Desintaller PICC8 HI-TECH
Editer la base de registre par menu démarrer -> executer -> regedit
Rechercher et effacer l'entrée HKEY_LOCAL_MACHINE\SOFTWARE\HI-TECH
Réinstaller PICC18 .... C18 fonctionne 20 jours
Sous WIN2000 ou XP il n'est utile de redémarrer l'ordinateur

Exécuter MPLAB

Dans le menu PROJECT cliquer INSTALL LANGUAGE TOOLS
Dans la suite HI-TECH PICC18 inscrire le même chemin d'installation de PICC18.EXE pour les trois outils Compiler, Assembler, Linker.



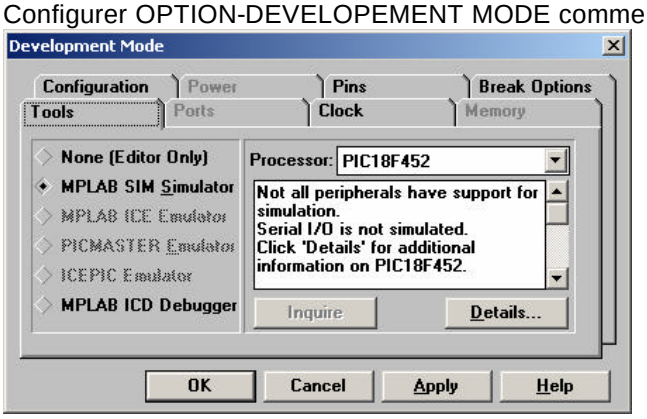
Le compilateur C inclut la notion de projet qui permet entre autre de compiler ensemble et avec des paramètres communs des fichiers d'origines différentes (C, ASM)
MPLAB offre la possibilité de ne compiler qu'un seul fichier source (ASM ou C) (ALT-F10) et donc de ne pas construire systématiquement un projet.

La librairie PIC18.h (et librairies associées pic18fxx2.h)
Elle contient toutes les définitions des registres et bits des PIC18.
Exemples de définitions dans pic18fxx2.h : @ retourne une adresse

```
static         near unsigned char    TRISC          @ 0xF94;
static         near unsigned char    TRISB          @ 0xF93;
static         near unsigned char    TRISA          @ 0xF92;
static         near unsigned char    LATC           @ 0xF8B;
static         near unsigned char    LATB           @ 0xF8A;
static         near unsigned char    LATA           @ 0xF89;
static volatile near unsigned char   PORTC          @ 0xF82;
static volatile near unsigned char   PORTB          @ 0xF81;
static volatile near unsigned char   PORTA          @ 0xF80;
#if defined(_18F442) || defined(_18F452)
static volatile near unsigned char   TRISE          @ 0xF96;
static         near unsigned char    TRISD          @ 0xF95;
static         near unsigned char    LATE           @ 0xF8D;
static         near unsigned char    LATD           @ 0xF8C;
static volatile near unsigned char   PORTE          @ 0xF84;
static volatile near unsigned char   PORTD          @ 0xF83;
#endif
//Latch B LATB
static         near bit       LB0            @ ((unsigned)&LATB*8)+0;
static         near bit       LB1            @ ((unsigned)&LATB*8)+1;
static         near bit       LB2            @ ((unsigned)&LATB*8)+2;
static         near bit       LB3            @ ((unsigned)&LATB*8)+3;
static         near bit       LB4            @ ((unsigned)&LATB*8)+4;
static         near bit       LB5            @ ((unsigned)&LATB*8)+5;
static         near bit       LB6            @ ((unsigned)&LATB*8)+6;
static         near bit       LB7            @ ((unsigned)&LATB*8)+7;
//Alternate definitions
static         near bit       LATB0          @ ((unsigned)&LATB*8)+0;
static         near bit       LATB1          @ ((unsigned)&LATB*8)+1;
static         near bit       LATB2          @ ((unsigned)&LATB*8)+2;
static         near bit       LATB3          @ ((unsigned)&LATB*8)+3;
static         near bit       LATB4          @ ((unsigned)&LATB*8)+4;
static         near bit       LATB5          @ ((unsigned)&LATB*8)+5;
static         near bit       LATB6          @ ((unsigned)&LATB*8)+6;
static         near bit       LATB7          @ ((unsigned)&LATB*8)+7;
```

1ier Programme : prise en main du compilateur

Créer un nouveau fichier (File-New)
Configurer OPTION-DEVELOPEMENT MODE comme suit



Créer un nouveau fichier premier.C comme suit :

```
/* premier programme en C*/
#include <pic18.h>
#define toutsort 0
#define toutentre 255

char a,b=3;

void main(void)
{
      TRISB=toutsort;
      TRISA=toutentre;

      a=PORTA;
      b=a+b;
      PORTB=b;
}
```

Fichier d'équivalences noms ✍ adresses

Equivalences

2 variables de type octet dont une initialisée à 3

Programme principal (main)

TRISA, TRISB, PORTA et PORTB sont déclarés dans pic18.h

Le PORTA est ajouté au PORB

Taper ATL-F10 pour compiler le programme dans la fenêtre sélectionnée

Infortional messages : verbose ✍précisera l'emplacement des erreurs
Generate debug info ✍permet un debug symbolique

Après compilation la fenêtre BUILD affiche :

```
Building PREMIER.C...

Command line: "C:\HTSOFT\PIC18\BIN\PICC18.EXE -V -ASMLIST -18F452   PREMIER.C"
Enter PICC18 -HELP for help

Memory Usage Map:
Program ROM   $000000 - $000003  $000004 (       4) bytes
Program ROM   $000006 - $000015  $000010 (      16) bytes
Program ROM   $000018 - $00006F  $000058 (      88) bytes
                                 $00006C (     108) bytes total Program ROM
RAM data      $0005FE - $0005FF  $000002 (       2) bytes total RAM data
ROM data      $000004 - $000004  $000001 (       1) bytes total ROM data

Program statistics:
Total ROM used      109 bytes (0.3%)
Total RAM used        2 bytes (0.1%)   Near RAM used        0 bytes (0.0%)
Build completed successfully.
```

Le compilateur a crée un programme de 109 octets …

Le compilateur place le haut de la RAM en 0x5FF
a est en 0x5FF
b est en 0x5FE

Les données non initialisées sont mises à 0 par la fonction clear_big en 0018
Les données initialisées le sont par la fonction copy_big en 0024

Recompiler an activant p*roduce assembler file list* ✍produit un fichier assembleur premier.as

Premier.as (partiel):

```
       psect   bigdata        secteur RAM de stockage des données initialisées
_b:
       ds      1

       psect   ibigdata       secteur ROM des valeurs d'initialisation
       db      3

       psect   text           secteur programme
_main:
;PREMIER.C: 11: TRISB=0;
       clrf    3987,c
;PREMIER.C: 12: TRISA=255;
       setf    3986,c
;PREMIER.C: 14: a=PORTA;
       movff   3968,_a ;volatile
;PREMIER.C: 15: b=a+b;
       movff   _a,wreg
       movlb   _b shr (0+8)   sectionne la bank de b (en ne gardant que les pf de l'adresse)
       addwf   _b & (0+255),f,b       ?
;PREMIER.C: 16: PORTB=b;
       movff   _b,3969 ;volatile
;PREMIER.C: 17: }
       goto    start

       psect   bigbss         secteur RAM de stockage des données non initialisées
_a:
       ds      1
wreg   equ     0xFE8
```

Mise au point : effectuer un RESET du processeur virtuel. Placer côte à cote les fenêtres C et Program memory. Placer un point d'arrêt sur main dans la fenêtre C. En mode pas à pas on visualise le déroulement du programme C et de l'assembleur correspondant.

Les options de compilation

Seules les options utilisées en développement sous MPLAB sont décrites ici

DANS COMPILER		
Produce assembleur liste file	-ASMLIST	Génère un fichier assembleur .LST (utile parfois pour le debug)
Error file	-E	Ecrit les erreurs dans un fichier (désactiver)
Informational messgaes	-V	Details les résultats de compilation (activer)
Generate Debug Info	-FAKELOCAL	Pour MPLAB
Assembleur optimisation	-O	Active une passe d'assemblage supplémentaire et peut parfois réduire le code
Include search path	-Ichemin,	Pour les fichier header qui ne sont pas dans include
Floating point for doubles	-D24 ou –D32	Précision des nombres réels 24 ou 32 bits. 32 bits est plus précis mains les calculs sont plus longs
Chars are signed	-Signed_char	Le type caractère est signé
Compile to assembleur code	-S	Comme ASMLIST mais sans être assemblé (fichier source)
Global optimisation	-Gx	1<x<9 (3 est un bon compromis) Permet d'optimiser la taille du code (9 est le plus performant mais le plus long)
DANS LINKER		
Create MAP file	-M	Affiche le plan mémoire
Display complete memory usage	-PSECTMAP	Affiche le plan mémoire et les secteurs
HORS SELECTION		
	-C	Compile vers un fichier objet seulement (pour création de bibliothèque)

Applications sur PICDEM2+

2ième Programme : gestion des ports parallèles

Créer un nouveau fichier avec le programme « bouton.c » ci dessous

```
/* La LED sur PB0 s'éteind si S2 (PA4) est enfoncé*/
#include <pic18.h>

void main(void)
{      TRISB = 0;            /* PB =      sortie */
       while(1)
      {
            if (PORTA&0x10) PORTB=1;
            else PORTB=0;
      }
}
```

Remarque : seule la LED sur PB devant être modifiée, on aurait pu écrire : LB0=x (à la place de PORTB) pour ne modifier que celle ci.

Très souvent les masques sont utilisés en C pour les tests ou les positionnements de bit

Ex pour tester si PA4=1

PORTA	x	x	x	x	x	x	x	x
&	0	0	0	1	0	0	0	0
=	0	0	0	x	0	0	0	0

Le résultat est nul si PA4=0. *Le C associe dans les tests la notion de faux au 0 et la notion de vrai à un nombre différent de 0.*

Ex positionner PA4 à 0

PORTA	x	x	x	x	x	x	x	x
&	1	1	1	0	1	1	1	1
=	x	x	x	0	x	x	x	x

Ex positionner PA4 à 1

PORTA	x	x	x	x	x	x	x	x
OU	0	0	0	1	0	0	0	0
=	x	x	x	1	x	x	x	x

PIC18.h possède des définitions de bits ce qui permet également de les tester ou les positionner directement :
LB0=0 ; ou LB0=1 (LB pour latch B et RB pour la broche B)

If (RA4) … ; else …. ; L'expression sera vraie si LA4 est non nul, il est donc inutile d'écrire (LA4==1)

Exercice : Modifier ce programme afin d'incrémenter PRB à chaque pression sur RA4.

3ième Exercice : Création d'une fonction

Recopier le programme led.c

```
#include <pic18.h>
#define duree 10000

void tempo(unsigned int count);

void main(void)
{
      PORTB = 0x00;
      TRISB = 0x00;
      while(1) {
            PORTB++;
            tempo(duree);
            }
}


void tempo(unsigned int compte)
{
      while(compte--);
}
```

La déclaration d'un prototype est nécessaire car la fonction tempo est définie après son appel

Boucle infinie incrémentant PRB

La fonction tempo reçoit un paramètre (int) qui est recopié dans la variable « compte », locale à la fonction. (duree n'est pas modifié)

Remarque : Si une fonction est écrite avant son appel le prototype devient inutile.

Exercice : modifier le programme led.c de manière à modifier la tempo (passer de 10000 à 20000) si S2 est appuyé.

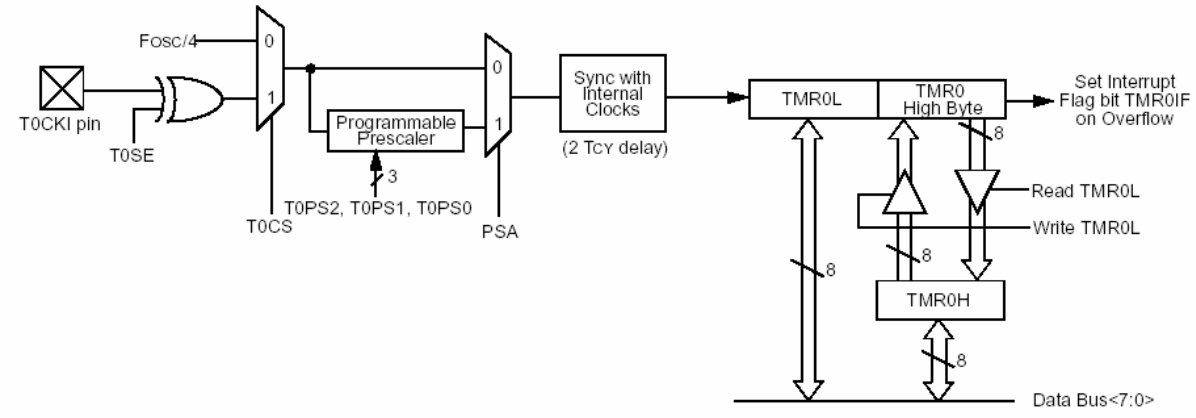
Exercice : Réaliser un programme faisant clignoter RB0 avec une période proche de 1s et un rapport cyclique ¼ si S2 est appuyé et ½ sinon.

4ième Programme : Interruptions

Programme flashit.c

```
// FLASHIT.c flash de la LED sur PB0 par it T=1.048s (TIMER0 produit des
temps de 2expN)
// Le débordement de TIMER0 provoque une it toutes les 524mS
#include <pic18.h>
#pragma interrupt_level 1     // choix du niveau d'interruption
void interrupt ITTIMER0()     //interrupt précise qu'il s'agit du SP d'it
{
      if(TMR0IF)              //vérifie un débordement sur TMR0
                  {
                  TMR0IF = 0; //efface le drapeau d'IT
                  LB0 = !LB0; //bascule LED sur RB0
                  }
}
void main()
{
      T0CON = 0x82;     //Active les TIMER0 - prescaler 1:8 -mode 16bits
      LB0 = 0;                // port B0 en sortie
      TRISB = !1;             // et à 0
      TMR0IE = 1;             //Autorise interruption sur TMR0
      GIEH = 1;               //autorise toutes les IT démasquées
      while(1);               // une boucle infinie, tout fonctionne en IT
}
```

Exercice : Modifier flashit.c de manière à obtenir une période de clignotement 4 fois plus longue.



REGISTER 10-1: T0CON: TIMER0 CONTROL REGISTER

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0
bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter
bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)
bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin
bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.
bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 prescale value 011 = 1:16 prescale value
110 = 1:128 prescale value 010 = 1:8 prescale value
101 = 1:64 prescale value 001 = 1:4 prescale value
100 = 1:32 prescale value 000 = 1:2 prescale value

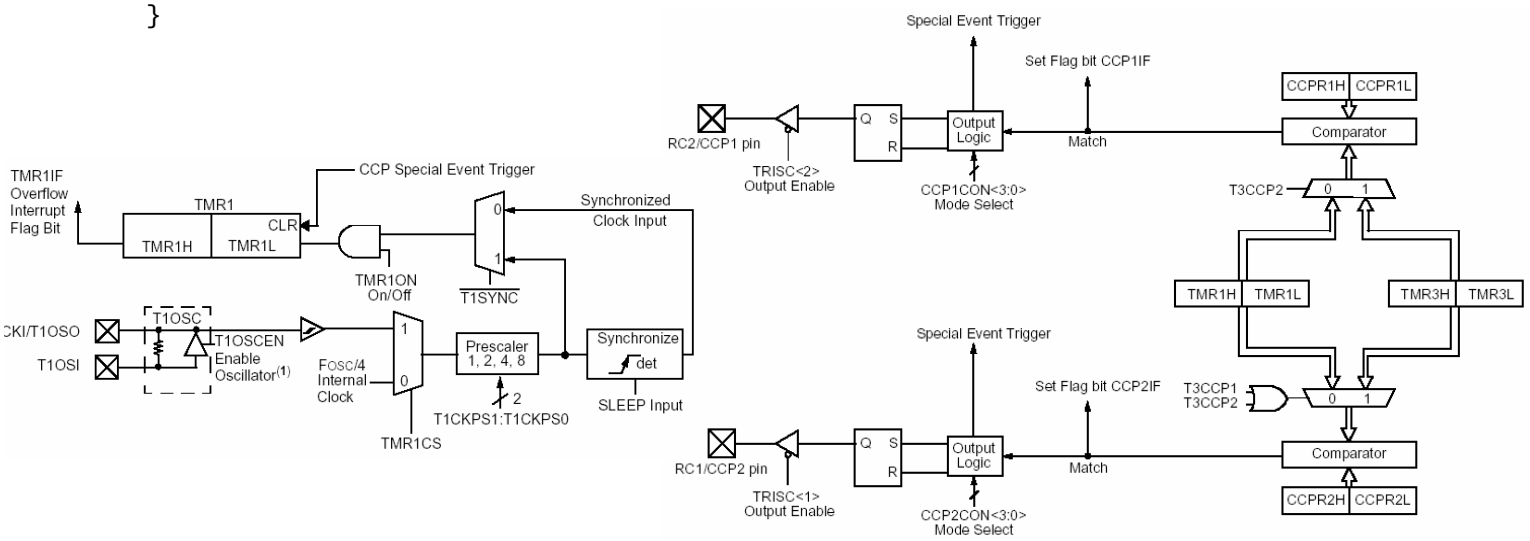
5ième Programme Production de temps

Le programme flashit.c ne permet pas de produire une durée de 500mS, pour cela il est nécessaire d'utiliser la fonction COMPARE associée au TIMER1 par exemple .

Programme itcomp.c

```
// FLASHIT.c flash de la LED sur PB0 par it T=1s (TIMER1 et  compare)
#include <pic18.h>
#pragma interrupt_level 1     // choix du niveau d'interruption
void interrupt versadresse18() //interrupt pour un SP d'interruption
{
static char tictac;     // IT toutes les 125mS, 4 IT avant de basculer PB0
      if(CCP1IF)              // l'IT provient d'une comparaison
                  {
                  if (++tictac==4) {
                                    LB0=!LB0;   //bascule PB0
                                    tictac=0;
                                  }
                  }
      CCP1IF=0;   //efface le drapeau d'IT
}

void main(void)
{
// configure PORTB
      LB0=0;            // RB0 en sortie
      TRISB0=0;
// configure le TIMER1
      T1RD16=0;   // TMR1 mode simple (pas de RW)
      TMR1CS=0;   // compte les impulsions sur internal clock
      T1CKPS1=1;  // prédiviseur =1/8 periode sortie = 8uS
      T1CKPS0=1;
      T1SYNC=1;   // pas de synchronisation sur sleep/Reset
      TMR1ON=1;   // TMR1 Activé
// configure le mode comparaison sur le TIMER1 avec IT sur CCP1 toutes les
15625 périodes de // 8us soit 125ms
      T3CCP2=0;   // mode comparaison entre TMR1 et CCPR1
      CCP1CON=0x0B; // Trigger special event sur comparaison
      CCPR1H=0x3d;      // égalité après 15625 périodes
      CCPR1L=0x09;
      CCP1IE=1;   // active IT sur mode comparaison CCP1
      IPEN=1;           // Interruption prioritaires activées
      GIE=1;            // Toutes les IT démasquées autorisées
      while(1);   // une boucle infinie, tout fonctionne en IT
}
```

Exercice : Modifier flashit.c de manière à obtenir un rapport cyclique ¼ sur S2 est enfoncé.



6ième Programme Afficheur LCD

La fonction putch(char) ou putchar(char) n'est pas définie en C. Elle doit être écrite en fonction du matériel afin d'écrire un caractère ASCII sur le terminal de sortie (afficheur LCD ou port série asynchrone)
Les fonctions puts(char *) et printf utilisent putchar pour écrire sur le terminal de sortie
Les bibliothèques lcdpd2.h et lcdpd2.c contiennent toutes les fonctions permettant de configurer l'afficheur lcd d'une carte PICDEM2+ et la fonction putchar.
Les bibliothèques delaypd2.c et delaypd2.h contiennent des temporisations utilisées par lcdpd2.c

void lcd_init(unsigned char); 0 pour FOURBIT_MODE et 1 pour EIGHTBIT_MODE
void lcd_cmd(unsigned char); envoie le caractère comme une commande
void lcd_putch(x); envoie le caractère comme une donnée
void lcd_puts(const char * s); envoie la chaine ASCII s sur LCD
void lcd_home() ; place le curseur à l'adresse 0 de l'afficheur
void lcd_clear() ; efface l'afficheur
void lcd_cursor(x) ;place le curseur à l'adresse x
void lcd_goto(x) ; idem
void lcd_cursor_right() ; déplace le curseur vers la droite lors d'une entrée
void lcd_cursor_left() ; déplace le curseur vers la gauche lors d'une entrée
void lcd_display_shift() ;curseur immobile, l'affichage se déplace vers la droite lors d'une entrée

Afficheur LCD : Adresses curseur
0x00 à 0x0F

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

0x40 à 0x4F

CODE ASCII (American Standard Code for Information Interchange)

code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	NP	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Exercices :
A partir de flashit.c et lcdtst.c, construire une horloge affichant heures, minutes, secondes. La mise à l'heure se fera dans le débogueur MP-LAB Dans un deuxième temps la mise à l'heure se fera par S2 pour les minutes et S3 pour les heures.

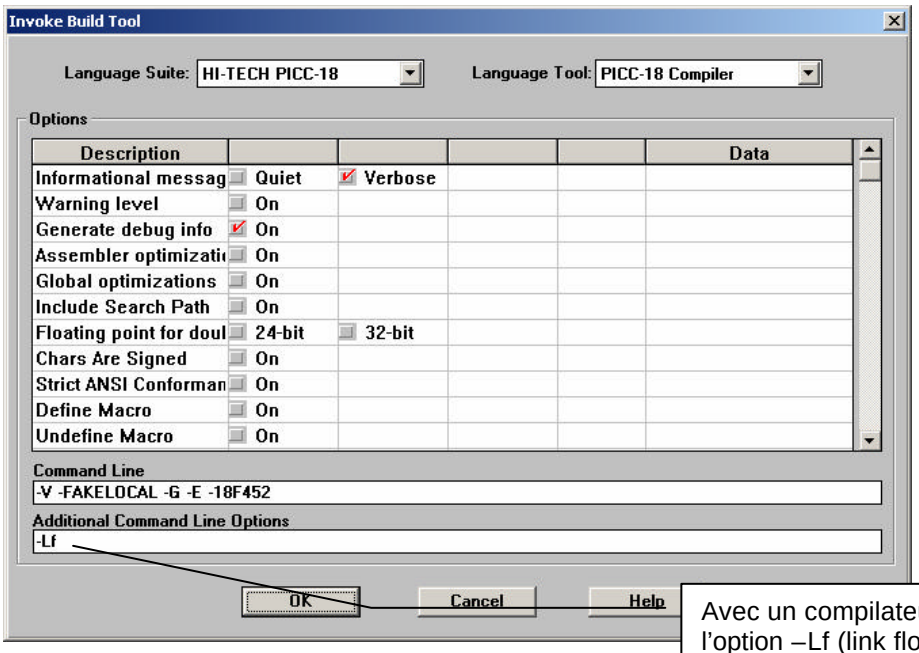
Programme lcdtst.c

```
#include <pic18.h>
#include "delaypd2.c"
#include "lcdpd2.c"
#include <stdio.h>

char chaine[]="abc";
void main(void)
{
      lcd_init(FOURBIT_MODE);
      lcd_cursor_right();
      lcd_cursor(0);                // positionne le curseur
      putchar('A');                 // un caractère
      lcd_cursor(5);
      lcd_puts("Bonjour");    // une chaîne, !!!puts n'est pas defini !!!
      lcd_cursor(0x40);
      printf("dec=%d hex=%x %s",10,10,chaine); // test partiel de printf
      while(1);
}
```

7ième programme : Nombres réels

La version « démo » du compilateur HI-TECH ne permet pas d'afficher des nombres réels avec printf
Le programme tstfloat.c contourne ce problème.

```
// Test fonctions et affichage sur les réels
// sans l'option %f de printf  test de cast
// on calcul f=5/3, on affiche ensuite deux entiers e et d
// e est la partie entiere de f et d la partie décimale
#include <pic18.h>      // équivalences générales pour PIC18
#include <stdio.h>      // pour printf
#include "delaypd2.c"   // tempos pour l'afficheur lcd
#include "lcdpd2.c"     // fonctions de base pour l'afficheur lcd

void main(void)
{
float f;
int e,d;

      lcd_init(FOURBIT_MODE);
      lcd_cursor_right();
      lcd_cursor(2);
      f=5.0/3.0;
      e=(int)f;
      d=(int)((f-(float)e)*10000.0);
      printf("5/3= %d,%d",e,d);
      while(1);
}
```



Avec un compilateur sous licence il faut utiliser l'option –Lf (link float) pour charger la librairie printf pour les réels ou le préciser dans les options de link

Exercice : modifier tstfloat.c, et introduire une fonction float to ascii : char* ftoa(float f,char *p) qui convertit un réel f en une chaîne de caractères p, la fonction retourne l'adresse de la chaîne. (on calcule deux entiers représentant les parties entières et décimales, puis l'on formate une chaîne avec sprintf
Exercice avec la bibliothèque mathématique math.h
Tester les fonctions sin, cos, sqrt … avec printf

8ième Programme : Créer une bibliothèque

Le programme ftoa.c montre l'utilisation d'une bibliothèque ftoabib.c
Le bichier ftoabib.c ne contient QUE la fonction char* ftoa(float f,char *p) créée dans l'exercice précédent. (Ce programme fera exactement la même chose que le précédent MAIS avec une nouvelle bibliothèque) Ici il n'est pas besoin de header puisque la bibliothèque n'est pas pré-compilée

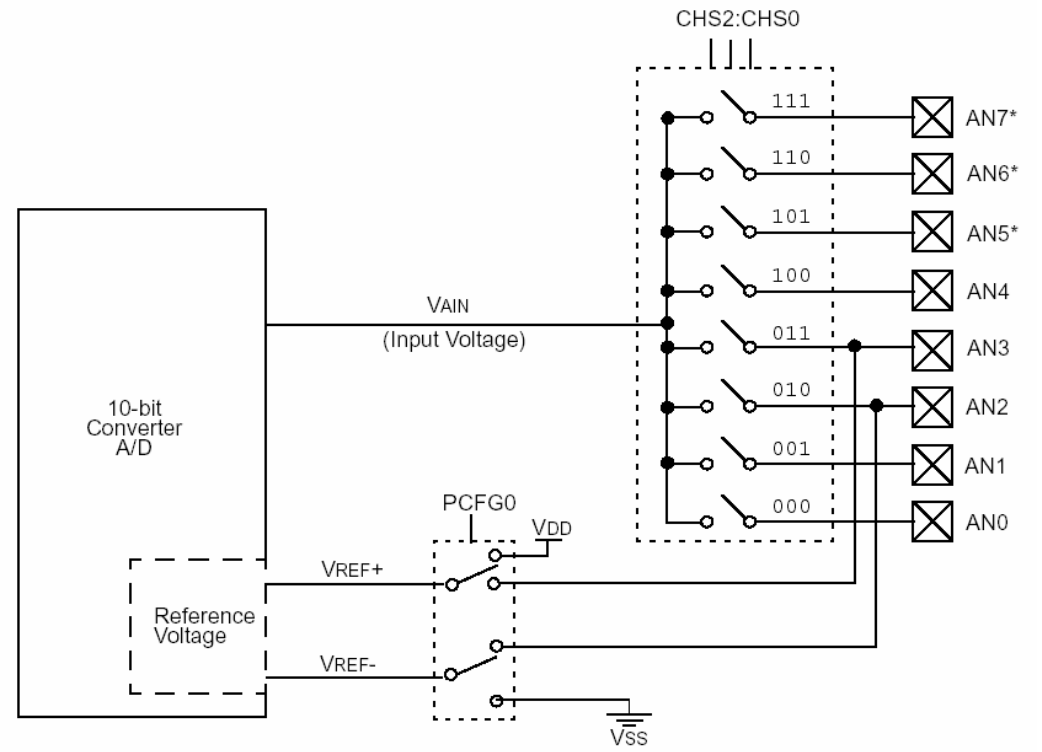
```
// CD 11/2002
// test de la fonction ftoa (float to ascii)
// pour palier la limitation des nombres réels sur printf dans la version
démo


#include <pic18.h>      // équivalences générales pour PIC18
#include <stdio.h>      // pour printf
#include "delaypd2.c"   // tempos pour l'afficheur lcd
#include "lcdpd2.c"     // fonctions de base pour l'afficheur lcd
#include "ftoabib.c"    // nouvelle bibliothèque contenant ftoa


char chaine[10];

void main(void)
{
float res;
      lcd_init(FOURBIT_MODE);
      lcd_cursor_right();
      lcd_cursor(2);
      res=5.0/3.0;
      ftoa(res,chaine);       // ftoa convertit le float res en une chaine
      printf("5/3= %s",chaine);
      while(1);
}
```

Déclaration de la nouvelle bibliothèque

9ième Programme : Conversion analogique/Numérique

Programme atod.c

```
// CD 11/02
#include <pic18.h>      // équivalences générales pour PIC18
#include <stdio.h>      // pour printf
#include "delaypd2.c"   // tempos pour l'afficheur lcd
#include "lcdpd2.c"     // fonctions de base pour l'afficheur lcd
#include "ftoabib.c"

void main(void)

{
      lcd_init(FOURBIT_MODE);
      lcd_cursor_right();
      ADCON0=1;               // CAN on. CLOCK=FOSC/2. CANAL0 (RA)
      ADCON1=0x8E; // justification droite, AN0 actif, VREF+=VDD VREF-=VSS

      while(1){
            GODONE=1;               // SOC
            while(GODONE);          // attend EOC
            lcd_cursor(1);
            printf("AN0= %d   ",ADRES);
            }
}
```

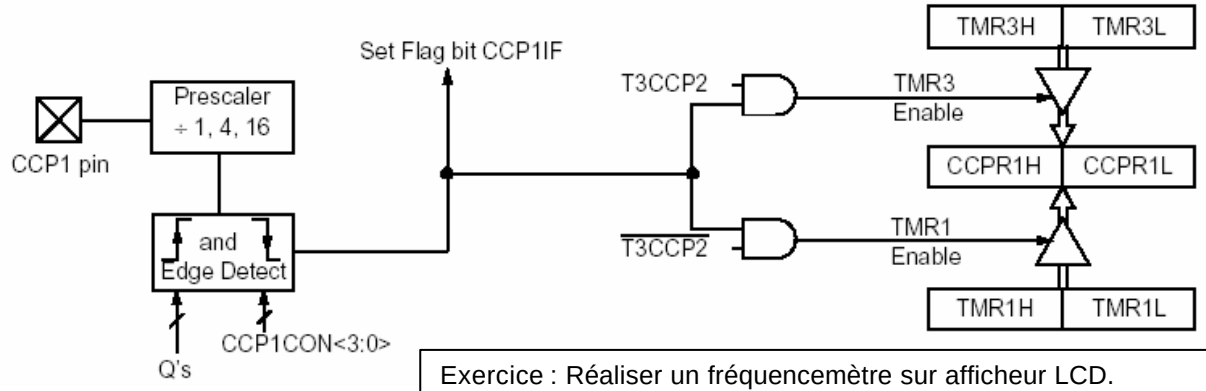


Exercice : Réaliser un voltmètre affichant la tension sur AN0 en volts en introduisant une fonction int mesvolt(char canal) retournant la valeur mesurée sur l'entrée canal

10ième Programme : Mesure de temps

Ce programme est similaire au programme Production de temps

```
// CD 11/02
// mesure de période sur CCP1 (RC2) (TIMER1 et fonction capture)
#include <pic18.h>

int duree;
char maj=1;

// sous programme d'interruption
#pragma interrupt_level 1

void interrupt versadresse18()
{
static char ancien;
      if(CCP1IF)             // l'IT provient d'une capture
                  {
                  duree=CCPR1H-ancien; // calcul nb impulsions comptés
                  ancien=CCPR1H;
                  maj=1;
                  }
      CCP1IF=0;   //efface le drapeau d'IT
}

void main(void)
{
// configure PORTB
      TRISC2=1;         // RC2 en entree (broche de CCP1)
// configure le TIMER1
      T1RD16=0;   // TMR1 mode simple (pas de RW)
      TMR1CS=0;   // compte les impulsions sur internal clock
      T1CKPS1=1;  // prédiviseur =1/8 periode sortie = 8uS
      T1CKPS0=1;
      T1SYNC=1;   // pas de synchronisation sur sleep/Reset
      TMR1ON=1;   // TMR1 Activé
// configure le mode capture sur le TIMER1 avec IT sur CCP1
      T3CCP2=0;   // mode comparaison entre TMR1 et CCPR1
      CCP1CON=0x05;     // capture mode sur fronts montants

      CCP1IE=1;   // active IT sur mode capture/comparaison CCP1

      IPEN=1;           // Interruption prioritaires activées
      GIE=1;            // Toutes les IT démasquées autorisées
      while(1) ;        // plus rien à faire
}
```



Exercice : Réaliser un fréquencemètre sur afficheur LCD.
Définir en fonction des paramètres de TIMER1 les fréquences max et min mesurables.

11ième Programme : Communications séries asynchrones

Programme tstusart.c

```
// CD 11/02
// Test des communications asynchrones avec IT en reception
// connecter un émulateur de terminal sur le pour série de PICDEM2+
// Attention au cable PC (brochage RX/TX)
#include <pic18.h>
#include <stdio.h>
volatile bit DATA_RECUE;            /* Indique qu'une donnée a été recue*/
volatile char DATA;                    /* Donnée recue */
char c;
void interrupt ISR(void)     // reception d'une interruption
{
      if((RCIE)&&(RCIF))           /* detection IT USART en reception*/
      {
            DATA=RCREG;
            DATA_RECUE=1;          /* signale qu'une donnée a été recue*/
      }
}
void putch(unsigned char c) //putch est défini sur le port série
{
    TXREG=c;                 /* envoie un caractère */
    while(!TXIF);
}
void main(void)
{
    GIEH=1;                  /*autorise les IT */
    GIEL=1;
    SPBRG = 25;              /* configure la vitesse (BAUD) 9600*/
    TXSTA = 0x24;
    RCSTA = 0x90;              /* active l'USART*/
    RCIE=1;                  // IT en reception activées
    DATA_RECUE=0;
    printf("\n Les communications sont ouvertes\n");
    printf(" --------------------------------\n");
    prints(" Tapez une touche ... ");
    while(1)
    {
      if (DATA_RECUE)  // on la renvoie en echo
        {
           printf("%c",DATA);
           DATA_RECUE=0;
        }
    }
}
```

Exercice : réaliser un voltmètre sur PC mesurant de tension sur AN0 et la transférant sur USART lors de la réception du caractère 'v'

Usart.c permet des tester les communications séries avec et sans IT.

Bibliothèque usartlib.c

Cette bibliothèque contient toutes les fonctions de gestion de l'USART
Configuration 9800,n,8,1

void Initsci(void) : configuration BAUD et interruption
char getsci(void) // retourne le premier caractère reçu sur SCI
void putsci(char c) // émet c sur SCI
char *getstsci(char *s, char finst) // lit une chaîne de caractère sur SCI se terminant par finst
// la variable finst contient le caractère attendu en fin de chaîne
int putstsci(char *s) // émet la chaîne s (finit par 0)