Prérequis : Notions de mémoires, de processeurs, d'algorithmique.

1. Rappels : Structure d'un uC, code machine et assembleur

Mémoire ROM (Read Only Memory) : Contient le programme codé en binaire et les données **constantes**. La mémoire ROM (souvent E2PROM) conserve les informations sans énergie.
Mémoire RAM (Random Acces Memory): Contient les données **variables** organisées en octets, elles peuvent être représentées par des nombres sur plusieurs octets. La mémoire RAM a besoin d'énergie pour conserver les informations.
ALU (Arithmétique and Logic Unit) : A partir d'instruction codées en binaire, l'ALU effectue des calculs arithmétiques et logiques et, déplace les variables entre les mémoires et les périphériques. L'ALU travaille sur des nombres de taille limité, un microcontrôleur 8bits effectue des opérations sur des nombres de 8bits (0 à 255 ou -128 à +127 si les nombres sont signés), le résultat étant lui aussi sur 8 bits.
Périphériques : Communiquent avec l'extérieur (ports // et séries), gère le temps (TIMER) , permettent d'interfacer des grandeurs analogique (ADC, DAC)…

Le **programme** est chargé de manière définitive dans la mémoire **ROM** par un l'outil de développement (ex : MPLAB + ICD3 chez Microchip). Lors du **RESET** du microcontrôleur, le **programme démarre** (ou redémarre). Il ne s'arrêtera jamais, un microcontrôleur ne fonctionne qu'en exécutant un programme, il y aura donc toujours une boucle sans fin dans un programme pour microcontrôleur.

Certaines données (variables) peuvent avoir à être initialisées à une valeur avant le lancement du programme, ainsi que certaines fonctionnalités du microcontrôleur comme les piles utilisées lors de l'appel des sous-programmes. Ces taches sont réalisées par un petit programme appelé **C Run Time (CRT)** fourni par l'éditeur du compilateur et toujours exécuté avant le programme principal, c'est le CRT qui lancera le programme de l'utilisateur.

Les programmes et les données sont codés en binaire sous forme d'octets dans les mémoires. Le programme en code machine est une suite de nombres binaires (souvent présentés sous forme hexadécimale pour les rendre plus lisibles).
Voici un exemple de code machine d'un programme :

Program Memory

Address	00	02	04	06	08	0A	0C	0E
00C0	F000	EC75	F000	EC67	F000	D7FB	0012	0100
00D0	518A	258B	0100	6F8C	D7FF	0012	EE00	F000
00E0	0E0F	6AEE	62EA	D7FD	0012	0012	0203	FFFF

Exemple : L'instruction à l'adresse 00D0 est 518A. Cette instruction effectue le chargement de la donnée contenue dans la case mémoire 8A vers le registre de travail du processeur. Ce type de représentation d'un programme est complètement opaque pour le développeur.
L'outil de développement logiciel de plus bas est niveau est basé sur le langage assembleur. Chaque fondeur de processeur développe son propre code machine et donc son langage assembleur.
En assembleur Microchip l'instruction à l'adresse 00D0 s'écrit `MOVF 0x8A,W` ce qui est à peine plus clair. L'assembleur est un langage mnémonique, MOVF signifie MOVE FILE (déplacer donnée) 0x8A étant l'adresse mémoire de la source (0x indique une base en hexadécimal) et W la destination. **Donc `MOVF 0x8A,W` signifie charger le registre W avec le contenu de la mémoire 8A.**

Le logiciel « assembleur » transforme le texte crée par le programmeur en code machines.

Exemple : adition du contenu de la case mémoire 0x8A à celui de la case mémoire 0x8B, le résultat étant placé en 0x8C (Les MOV sont les déplacements de données, ADDWF est l'instruction addition)

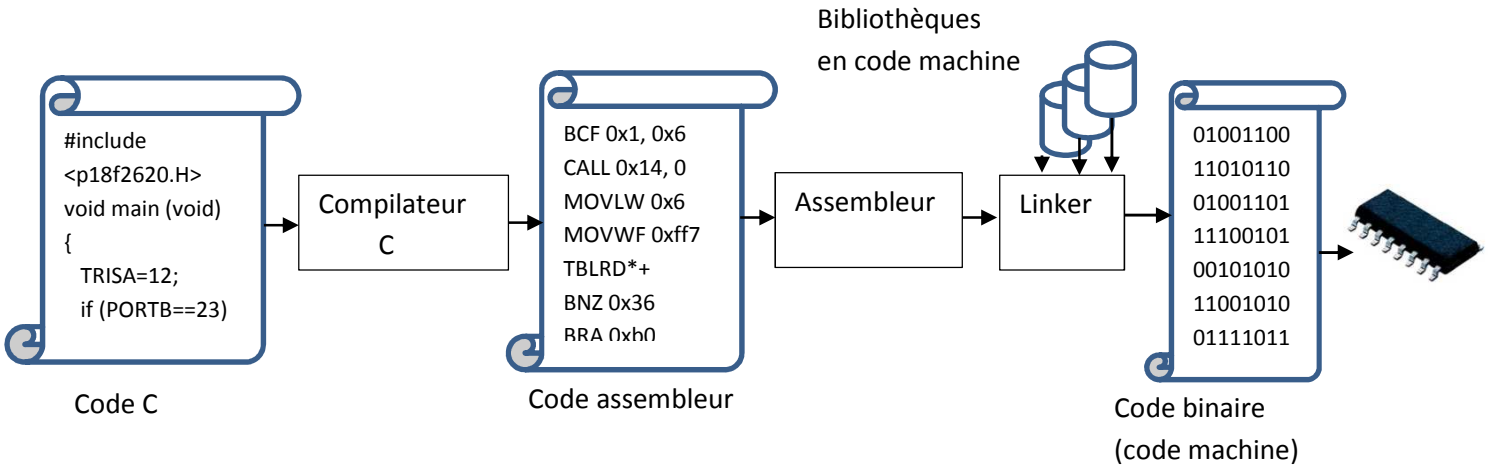
```
00D0    518A    MOVF 0x8a, W,
00D2    258B    ADDWF 0x8b, W
00D4    0100    MOVLB 0
00D6    6F8C    MOVWF 0x8c, B.
```

Organisation d'un programme dans la mémoire ROM (exemple MICROCHIP mémoire ROM de 64KO)	Adresses	Nb octets
RESET	0x0000	3
Sous-programme d'interruption prioritaire	0x0008	16
Sous-programme d'interruption non prioritaire	0x0018	116
C Run Time	Ox0028	98
Programme principal	Ox00C0	34
Sous-programme A	0x00F4	79
Sous-programme B	0x0143	993
Sous-programme C	0x0524	289
Constantes	0x0645	5
Vide	0x0650	63919
	0xFFFF	

C'est un logiciel appelé "**LINKER**" qui attribue les zones mémoires aux programmes et données.

2. Chaine de développement logiciel

Afin de faciliter la programmation et surtout de la rendre portable (c'est-à-dire identique pour toutes les cibles) des langages de haut niveau ont été créé. **Le langage « industriel » est le C.**



3. Langage C

Le langage C a été créé pour programmer les ordinateurs à des fins de traitement de données. De nombreuses fonctionnalités du langage ne sont pas ou peu utilisées dans le cadre de la programmation d'un microcontrôleur. En revanche le programme d'un microcontrôleur devra accéder aux périphériques et à leurs configurations.

a. Syntaxe des nombres et variables

128 en decimal
0x80 en hexadecimal
0b10000000 en binaire
} C'est le même nombre dans différentes bases de numération
'A' représente le code ASCII de A

En C toute donnée doit avoir été déclarée avant d'être utilisée. La déclaration indique la taille en octets de la variable ainsi que son étendue (toujours positive ou pas)

Tableau de déclaration (à bien connaitre), exemple pour un PIC18).

Type	Longueur	Domaine de valeurs	utilisation
char	8 bits	-128 à 127	Nombre entier signé
unsigned char	8 bits	0 à 255	Nombre entier positif ou code ASCII
int	16 bits	-32768 à 32767	Nombre entier signé
unsigned int	16 bits	0 à 65535	Nombre entier positif
long	32 bits	-2,147,483,648 à 2,147,483,647	Nombre entier signé
unsigned long	32 bits	0 à 4,294,967,295	Nombre entier positif
float	32 bits	3.4 * (10**-38) à 3.4 * (10**+38)	Nombre réel
double	64 bits	1.7 * (10**-308) à 1.7 * (10**+308)	Nombre réel
pointeur	16 bits	0 à 65535	Adresse d'une donnée

b. Accès aux données

```
char a;              // a est un octet
unsigned int b,c;    // b et c sont codés sur 2 octets
char *p;             //  p est l'adresse d'un octet
```

Les données sont accessibles par valeur :

```
a=21 ;
b=0x342
c=b ;
```

ou par adresse avec l'opérateur *.

Si p est l'adresse (un pointeur) d'un octet:

a=*p ; Se lit: a égale le contenu de l'adresse p (p est alors considéré comme une adresse de char)

L'opérateur & (ne pas confondre avec l'opération ET) permet d'accéder à l'adresse d'une donnée:

p=&a; Se lit : p égale l'adresse de la donnée a, cela suppose que b soit un pointeur (une adresse)

Les données peuvent être regroupées en tableau. Un tableau est un ensemble de données de même type. Le nom du tableau représente l'adresse du premier élément du tableau. Les éléments sont accessibles par un indice qui va de 0 à la taille du tableau -1.

```
char mon_tableau[12] ;  // Declare une zone mémoire de 12 octets non initialisée
char mes_donnees[]={23,56,0x12,'A'} ; // 4 octets initialisés
```

a=mes_donnees[2]; // a prend la valeur 0x12
mon_tableau[0]=a; // le premier élément du tableau prend la valeur 0x12 (celle de a]

Les tableaux sont souvent utilisés pour déclarer des chaines de caractères ASCII.
unsigned char mess[]= " bonjours tout le monde" ;
Les chaines de caractères (string en anglais) finissent toujours par un 0 ajouté automatiquement)
On peut alors écrire des fonctions comme *envoyer(mess);* mess étant l'adresse du message

Les équivalences ne génèrent pas de code machine mais rendre les programmes beaucoup plus lisibles:

```
PORTAbits.RA4=1;     ou PORTAbits.RA4=0;
```

(mais qu'est-ce que c'est ?)
peut être remplacé par

```
#define LED  PORTAbits.RA4
LED=1;  ou LED=0;     (c'est plus clair)
```

Les #define doivent être placées en dehors de toute fonction et en début de programme

En C toute instruction ou groupe d'instructions est dans un **sous-programme que l'on appelle fonction**, y compris le programme principal. Une fonction est un ensemble d'instructions groupées et encadrées par des accolades, le nom de la fonction représente l'adresse ou celle-ci est rangée en mémoire, pour exécuter une fonction il suffit d'écrire son nom.
Elle peut posséder un ou plusieurs paramètres d'entrée mais un seul paramètre de sortie, les types des paramètres sont déclarés comme les variables. S'il n'y a pas de paramètre on écrit **void** (vide).
Exemple : le calcul du sinus d'un angle de 45° rangé dans la variable "résultat" s'écrira

```
resultat=sin(45) ;
```

ou bien

```
angle=45 ;
resultat=sin(angle) ;
```

remarque : en C toute instruction se termine par ;

La déclaration de la fonction sin ci-dessus (lors de sa création) sera

```
float sin(float alpha) ;
```

sin reçoit un paramètre réel qui s'appellera alpha dans le corps de la fonction et retourne un résultat sous forme d'un nombre réel.

Autre exemple : allumer une led parmi huit:

```
void allume_LED (unsigned char num_LED) ;
```

allume_LED reçoit un octet qui contient le numéro de la LED à allumer mais ne renvoie rien, on l'utilisera comme cela :

```
allume_LED(3) ;    //pour allumer la LED numéro 3
```

c. programme 1 : Le programme minimum, il ne fait rien d'ailleurs …

```
void main (void)
{
     while(1)   {     }
}
```

Le programme principal est une fonction nommée "main" (mot réservé pour le point d'entrée du programme), elle n'a ni paramètre en entrée ni en sortie, ce qui normal puisqu'elle n'est pas appelée et possède une boucle sans fin (on ne sort jamais de main).
Le corps de la fonction est placé entre deux accolades. Les accolades encadrent également les groupes d'instructions dans les tests et les boucles (voir ci-dessous). La fonction main contient au moins une boucle sans fin. Elle s'écrit « tant que » « condition » est vrai. (while en anglais).
En C, 0 est « faux » » toute autre valeur est « vraie ».
while(1) { } est une boucle qui peut se lire : tant que « 1 » est vrai rester dans la boucle et exécuter son contenu. Il n'y a rien dans le corps de la boucle donc il n'y a rien à faire. Comme 1 ne sera jamais nul on ne sortira jamais de la boucle.

d. Programme 2 : création de variables

Cet exemple additionne deux octets (8 bits) et range le résultat dans un entier (16 bits)

```
unsigned char a=245,b;
int res;
void main (void)
{
      b=124;
      res=a+b;
      while(1)   {     }
}
```

Les variables doivent être déclarées avant d'être utilisée afin de leur réserver un emplacement dans la mémoire RAM. C'est le LINKER qui attribuera une espace mémoire à chaque variable. Ici un octet pour a et b et deux octets pour res.
La variable a est initialisée dans la déclaration, c'est le programme CRT qui chargera la case mémoire de a avec la valeur 245 avant de lancer le programme main.
Le C possède de nombreux autres opérateurs comme +, -, *, / etc ….

e. Programme 3 , les fonctions

Les fonctions doivent être déclarées avant d'être appelées. La fonction pourcent de ce programme calcule les x pourcents de la valeur y et les retourne dans un entier.

```
char val;
char res;

int pourcent(char x, char y)
{     int z;
      z=(y/100)*x;
      return (z);
}

void main (void)
{
      b=val;
      res=pourcent(12,val);
      while(1)   {     }
}
```

les paramètres 12 et val sont recopies respectivement dans les variables x et y, le calcul s'effectue sur x et y. x, y et z sont des variables " locales", elles sont créés dans une pile lors de l'appel de la fonction et détruites à la sortie.

f. Programme 4 test : if-else

Les tests permettent d'exécuter une instruction ou un groupe d'instructions (délimité pas des accolades) si la condition est vrai. Le "else" peut être facultatif.

```
if (a>5) b=4 ; // si a est supérieur à 5 b égale 4 sinon b égale 0
else b=0 ;
```

Comme le test s'effectue sur une condition vraie ou fausse on peut écrire

```
if (a !=0 )    ou  if (a)        si a est différente de 0
if (a==0)    ou    if( !a)          si a est égale à 0
```

IMPORTANT : le test d'égalité s'écrit ==, l'assignation =

g. Programme 5, boucles

Les boucles sont effectuées tant que la condition testée est vrai.

En C il existe 3 boucles, while, do-while et for.

```
while (a>5)       // si a est plus petit que 5 la boucle n'est pas effectuée
      {           a= PORTB ;
                 PORTA=PARTA++ ;
      }

do    {          a= PORTB ;
                 PORTA=PARTA++ ;
      }
while (a>5)       // si a est plus petit que 5 on sort de la boucle

for (i=0;i<10;i++)      // ici la boucle est effectuée 10 fois
      {          PORTA=PARTA++ ;
      }
```

h. Programme 6, C et microcontrôleurs

Le programme ci-dessous assigne la valeur du PORTB à la variable a. Cela n'est possible que si le compilateur connait l'adresse de PORTA et de PORTB. Les compilateurs sont toujours accompagnés de fichiers de déclaration appelés « header » avec l'extension .h

Les fichiers (.h) peuvent contenir les déclarations de fonctions précompilés en bibliothèques comme <stdio.h> ou <math.h>. Ils peuvent également contenir les déclarations de registres et bits d'un microcontrôleur.

Le fichier p18f2620.h présent ici contient les déclarations des tous les registres et bits du microcontrôleur P18F2620 de Microchip. Les noms des registres sont les mêmes que ceux du datasheet, les bits sont accessibles par le structure : NOMDUREGISTREbits.NOMDUBIT

ex : PORTAbits.RA0 représente le bit 0 du PORTA.

Les microcontrôleurs étant très versatiles, ils doivent être souvent préconfiguré avant d'exécuter le

programme. On trouve très souvent en haut du fichier des directives #pragma permettent la pré configuration du microcontrôleur, exemple pour un PIC :

```
#include <p18f2620.h>    // contient les declarations des registres et bits
#pragma config OSC = INTIO7   // active l'horloge interne
#pragma config PBADEN = OFF, WDT = OFF, LVP = OFF, DEBUG = ON, XINST=OFF

char val;
void main (void)
{     TRISA=0x00;            // PORTA en sortie
      TRISB=0xFF;            // PORTB en entree
      TRISCBits.TRSIC3=1; // Bit 3 du PORTC en entree
      while(1)
            {     PORTA=PORTB;
                  val=PORCbits.RC3; // val serai vraie ou fausse
            }
}
```

i. Programme 7, sorties de messages en texte.

Le C est accompagné de nombreuses bibliothèques (library en anglais) de fonctions qui ont été standardisées avec le langage. La plus connue est "stdio" pour *Standard Input Output*. Celle bibliothèque contient de nombreuses fonctions gérant les entrées et sorties de chaines de caractères ASCII. L'exemple le plus connu est le programme "hello word" que l'on trouve dans tous les livres sur le C.

```
#include <stdio.h>
void main (void)
{     printf("bonjour le monde");  // hello world mais en français
      while(1) {};
}
```

Une fois le #include effectué, toutes les fonctions de la bibliothèques sont disponibles (dont ici printf, qui envoie une chaine de caractères ASCII vers le périphérique de sortie)

Ce petit programme écrit le message "bonjour le monde" sur le périphérique de sortie. Sur un ordinateur il y a un écran mais sur un microcontrôleur il n'y a que le port asynchrone (appelé souvent UART ou USART). C'est donc sur ce port série que le message est envoyé. *Il est cependant toujours possible de réorienter le flux de données vers un autre périphérique interprétant le code ASCII*.
L'UART du microcontrôleur doit en général être configurée en fonction de la vitesse de communication voulue et de la base de temps interne (qui est la plupart du temps l'oscillateur du microcontrôleur). Le programme "hello word" pour un microcontrôleur PIC18F2620 s'écrira :

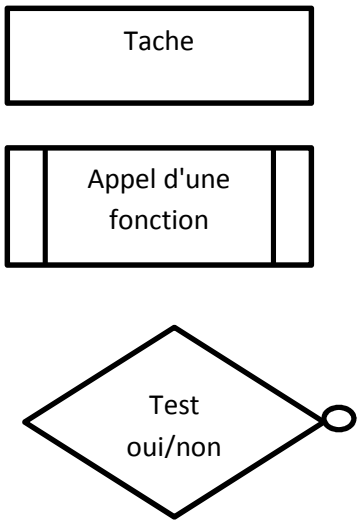
```
#include <p18f2620.h>
#include <stdio.h>
#pragma config OSC = INTIO7   // active l'horloge interne
#pragma config PBADEN = OFF, WDT = OFF, LVP = OFF, DEBUG = ON, XINST=OFF

void init_UART(void)
{   OSCCONbits.SCS1=1;   // oscillateur interne FOSC=1MHz
    BAUDCONbits.BRG16=1; // necessaire pour 9600Baud avec FOSC=1MHz
    SPBRG = 25;           // pour 9600 Bauds
    TXSTA = 0b00100100;  // active l'UART en sortie
    RCSTA = 0b10010000;   // active l'UART en entree
}
```

```
void main (void)
{     init_UART();
      printf("bonjour le monde");
      while(1) {};
}
```

Comme on le voit il sera nécessaire de configurer le microcontrôleur avant d'utiliser ses périphériques. Une bonne connaissance de la cible est indispensable pour la programmation en C embraqué .

4. Exercice 1 :



Interpréter ce programme et dessiner son algorigramme

```
#include <p18f2620.h>
#define led PORTBbits.RB0
#define S2 PORTAbits.RA4
#define duree1 10000
#define duree2 20000

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

void main(void)
{     TRISBbits.TRISB0 = 0;
                TRISAbits.TRISA4=1;
      while(1) {
                        led=!led;
                        if (S2)     tempo(duree1);
                        else tempo(duree2);
            }
}
```

5. Exercice 2 :

En vous aidant de ce cours et de ressources trouvées sur internet, analyser ce programme, indiquer la fonction de chaque ligne. Donner un exemple de valeur de PORTB pour PORTA=0x36;

```
#include <p18f2620.h>
#include <stdio.h>
void main(void)
{  TRISB=0;
   TRISA=0xFF;
   while(1)
          {   PORTB=(PORTA>>4)&0b00001111;
              printf("PORTA= %d    PORTB = %d  \r\n",PORTA,PORTB);
          }
}
```

Vous en savez déjà beaucoup, pour en savoir plus…

Consulter les très nombreux cours en ligne sur le C (http://genelaix.free.fr)

Lire les documentations des compilateurs. Programmer, c'est le meilleur moyen d'apprendre