Aix*Marseille université

Licence SPI - AGE

Travaux Pratiques
Langage C18

Julien Marot
julien.marot@fresnel.fr

Zouhair Haddi
zouhair.haddi@lsis.org

Marc Allain
marc.allain@fresnel.fr

Table des matières

1. Avant de commencer

Ces séances de travaux pratiques (TP) sont destinées à illustrer les notions qui vous ont été présentées en cours. Pour cela, vous aurez à concevoir et à tester un certain nombre de programmes en langage C.

Pour chaque tâche que vous aurez à réaliser, nous vous invitons 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 primordial que vous commenciez par rédiger un algorigramme, que vous commentiez clairement votre programme et que vous pensiez à le rendre le plus clair possible (*ex* : en choisissant des noms appropriés pour les labels).

Concernant l'évaluation de votre travail, nous vous demanderons de nous présenter le programme correspondant à chaque tâche que vous aurez à programmer sur le simulateur ou sur le kit de démonstration (*cf.* plus bas). Nous corrigerons également vos codes en portant une attention particulière à la lisibilité (pensez donc aux commentaires !). Enfin, vous devrez également nous rendre les algorigrammes qui correspondent aux codes.

2. Le Matériel

2.1. Le Kit PICDEM 2 Plus et PIC 18F4520

Le kit PICDEM 2 Plus est un kit simple qui permet de mettre en évidence les capacités de différents micro-contrôleurs du fabricant Microchip à travers divers périphériques et accessoires reproduis sur le schéma de la figure 1.

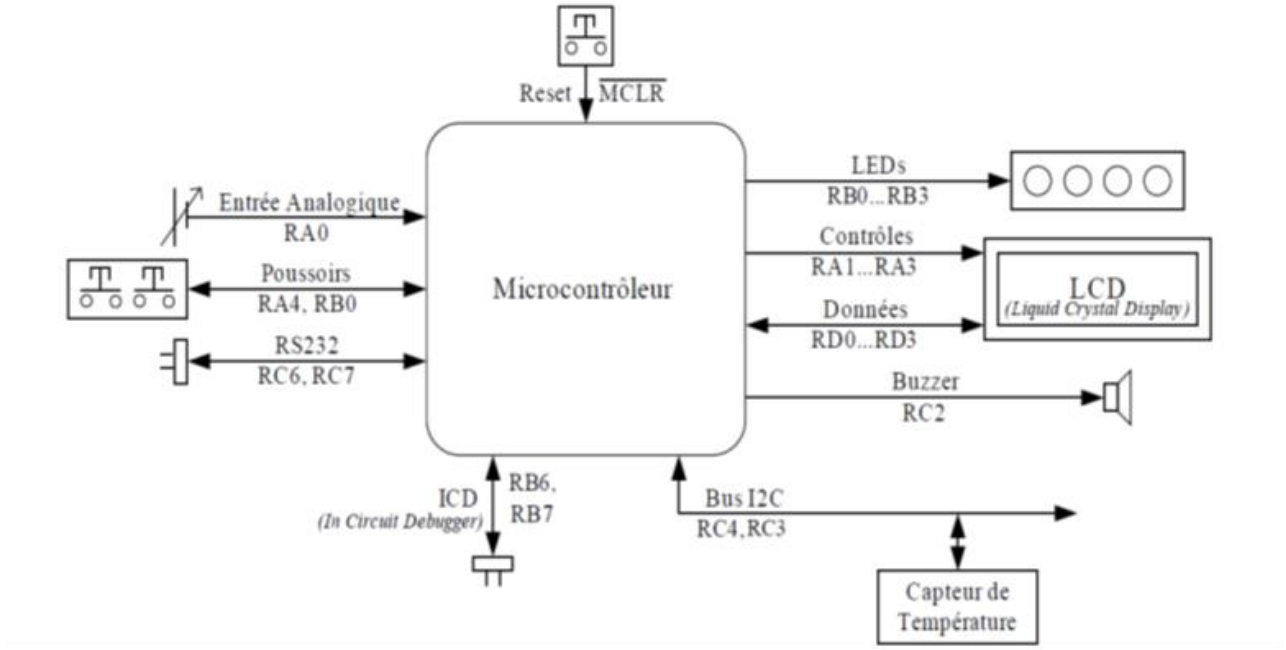

Figure 1: Le kit PICDEM 2 Plus, Périphériques et Accessoires

Le micro-contrôleur utilisé est le PIC18F4520 de Microchip. C'est un micro-contrôleur 8 bits disposant d'une mémoire de programme de 32 Ko, de 1536 octets de RAM (Random Access Memory), d'une EEPROM (Electrically Erasable Programmable Read Only Memory) de 256 octets et de 36 entrées/sorties.

2.2. MPLAB Integrated Development Environment

MPLAB IDE est un environnement gratuit de programmation pour les micro-contrôleurs de la famille Microchip. L'environnement intègre un éditeur d'assembleur, un débogueur complet ainsi que tous les outils permettant de gérer les programmateurs de composants. La possibilité d'intégrer un compilateur C dans l'environnement de développement MPLAB s'inscrit directement dans la politique du service qui est d'utiliser des langages de programmation évolués dans les développements en électronique.

2.3. Le programmateur ICD3

Il permet de transférer le programme directement de MPLAB dans le micro-contrôleur via un port USB d'un PC.

3. TP n°1 : Initiation à la programmation en langage C

3.1. Introduction : assembleur et compilateur C

L'objectif de cette séance est de vous initier à la *programmation en langage C* d'un micro-contrôleur. Par rapport à l'assembleur, le recours à un langage de « haut niveau » tel que le C s'est notablement développé en pratique car il permet de simplifier l'écriture des codes ainsi que leurs portabilités et leurs maintenabilités. Notamment, vous pourrez écrire des instructions du type « *if(x==y)* » ou « *temp = 0x27* » pour effectuer un test d'égalité ou une affectation. Le code que vous écrirez devra être écrit en C standard ANSI. La compilation des instructions C se fera par le compilateur C18 qui permet de générer du code pour la famille des PIC 18XXXX. Comme l'assembleur, le compilateur traduit des instructions interprétables pour un opérateur humain en un code machine exécutable par le PIC. Comme nous le verrons, l'écriture et la compilation de vos programmes sources C se fera dans l'environnement MPLAB. Comme pour le développement de codes en assembleur, MPLAB sera donc l'environnement qui vous permettra d'écrire, de compiler et de déboguer vos sources C.

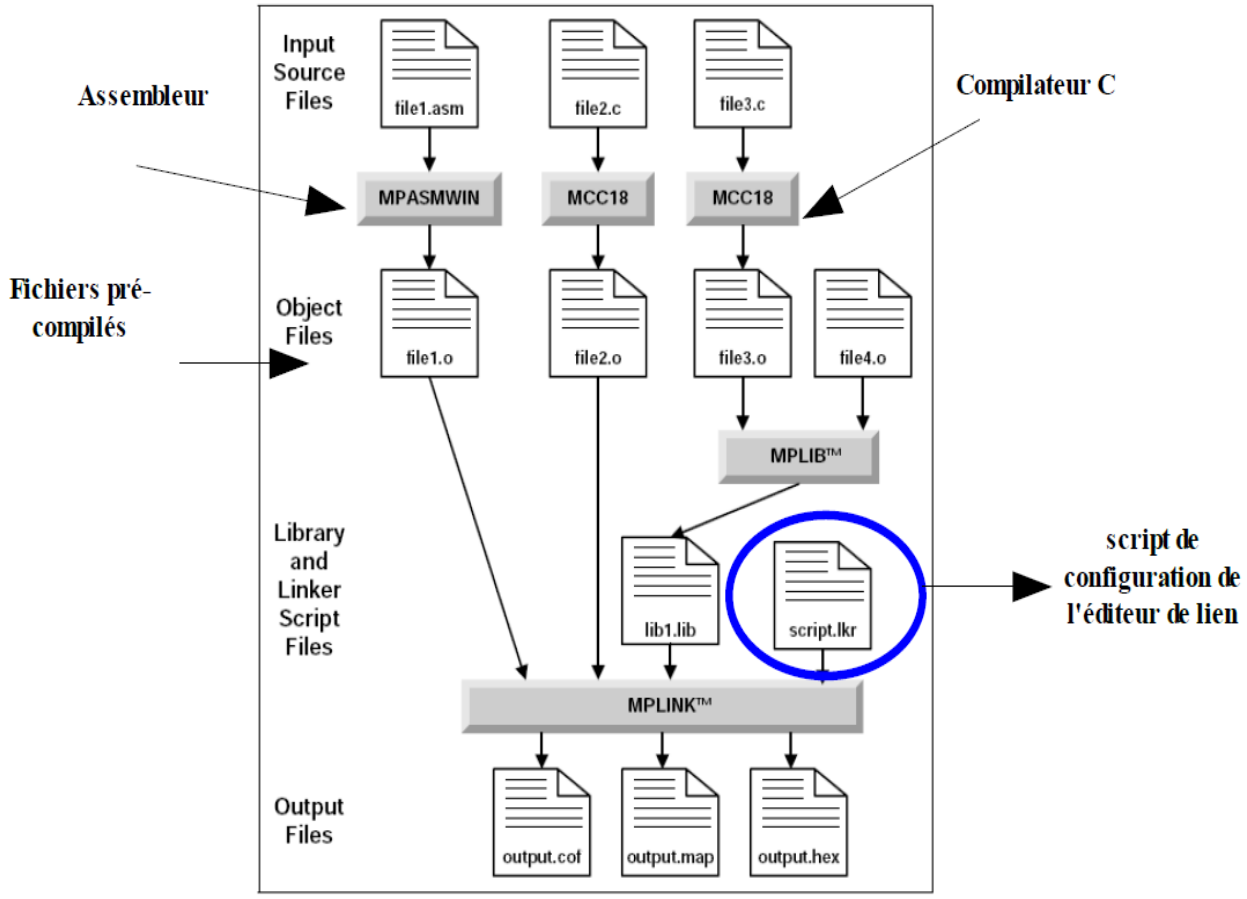

Figure 2: Diagramme de construction d'un exécutable

On notera que, dans le cadre d'un *même* projet, il est possible de développer des parties en C et des parties en assembleur. Dès lors, la construction d'un *code machine exécutable* est illustré par le diagramme de la figure 2. Dans cet exemple, un code source en assembleur et deux codes sources en C ont été écrits. Pour chaque source, l'**assembleur** (MPASMWIN) ou le **compilateur**(MCC18) est utilisé pour générer des *fichiers objets* (*file1.o,...,file3.o*). Le fichier objet *file3.o* est ensuite utilisé avec un fichier pré-compilé existant *file4.o* de manière à former un fichier de librairie *lib1.lib*. Finalement, les fichiers objets restants sont utilisés avec le fichier de librairie par l'**éditeur de lien** (MPLINK) pour créer un exécutable *output.hex*.
Notez bien également le fichier *script.lkr* (*cf.*, figure 2) qui permet de configurer correctement l'éditeur de lien pour le micro-contrôleur utilisé.

3.2. Génération d'un signal carré

Pour vous familiariser avec la programmation en C du micro-contrôleur, le *premier projet* sera de reprendre le programme de clignotement d'une LED que vous aviez écrit en assembleur. Le source C ci-après fait clignoter la LED L1, broche RB0, à une fréquence de 1Hz. L'état de la LED change donc toutes les 500ms.

En page suivante se trouve le programme en C dédié à la génération de ce signal carré et que vous devez écrire.

```
// Author:   ???
//================================================
// Description :
// Programme de génération d'un signal carré
// sur le port RB[0] de période de 1 s
//================================================


//------------------------------------------------
// Directives au préprocesseur
//------------------------------------------------
// fichier d'en-tête pour le PIC184020
#include <p18f4520.h>
// fichier d'en-tête pour les fonctions de tempo
#include <delays.h>

// Désactivation du WatchDog
#pragma config WDT = OFF


// Programme Principal
void main(){

//------------------------------------------------
// Déclaration des variables locales
//------------------------------------------------

// 1/2 Periode d'oscillation en Nb de cycles
TYPE DemiPeriode = ARGUMENT;

// Configuration du portB
TRISB = 0x00;                   // PORTB en sortie
PORTB = 0x00;                   // RAZ du PORTB
// Config. RB0-RB3 en mode I/O numerique
ADCON1 = 0x0F;


// --> Boucle TantQue : On boucle indéfiniement...
while(1){
      // basculement du bit 0 du PORTB
      PORTB = PORTB ^ 0x01;
      // Utilisation d'une fonction de delays.h
      FONCTION(ARGUMENT);
      }
// --> Fin de boucle Tant Que
}
```

En-tête du programme

Fichiers d'en-tête pour le PIC184520 (p18f4520.h) *et pour l'utilisation des fonctions de temporisation* (delays.h)

Directive « pragma » de désactivation du watchdog.

Déclaration et affectation de la variable définissant le nombre de cycles nécessaire à une demi-période d'oscillation (**à compléter**).

Configuration du port B

Boucle permanente de changement d'état de RB0
-> basculement de RB0
-> appel à la fonction de temporisation (cf. delays.h*) permettant d'attendre 500ms* (**à compléter**).

Comprendre le programme ci-dessus.

Définir le type de la variable *DemiPeriode*.

Déduire du fichier d'en-tête « *delays.h* » la fonction à utiliser pour une temporisation telle que la période soit de 1ms. Déduisez-en également l'argument associé.

Passer à la section suivante pour créer un projet pour ce code, pour le compiler et pour effectuer la simulation.

3.3. Édition et Compilation du programme

Cette partie permet d'apprendre à créer un projet MPLAB, à saisir un programme en C et à le compiler.

3.3.1. Démarrage

- Avant de commencer, créer le répertoire « *tp1* **»** puis dans ce répertoire créer à nouveau un répertoire « *clignotement_C* ».
- Copiez dans ce répertoire les fichier « *delays.h* » et « *18f4520.lkr* ».
- Lancer la suite logicielle MPLAB IDE à partir de l'icône qui se trouve sur votre bureau.

3.3.2. Création d'un nouveau projet en utilisant l'assistant « Project Wizard … »

- Cliquer sur *Project >> Project Wizard …*
- Cliquer sur *Suivant* > puis sélectionner « *PIC18F4520* » dans le menu déroulant « *Device* ».
- Cliquer sur *Suivant* > puis sélectionner « *Microchip C18 Toolsuite* » afin de pouvoir développer en C.
- Cliquer sur *Suivant* > puis dans le champ « *Project Name* » saisir le nom du projet (ex : clignotant_C) et dans le champ « *Project Directory* » aller chercher à l'aide du bouton « *Browse* », le dossier créé au début : « *tp1/clignotant_C* ».
- Cliquer **deux fois** sur Suivant > et ajouter « *18f4520.lkr* » au projet à l'aide du bouton *Add >>*
- Cliquer sur *Suivant* > puis sur *Terminer*.

3.3.3. Configurer les options de compilation et d'édition de lien

- Cliquer sur *Project>Build Options...>Project* et cliquer sur l'onglet « *General* ». Vérifiez que les chemins de « *Include Path* » et de « *Library Path* » sont correctement positionnés en utilisant l'option « *Browse* »...
- Dans l'onglet MPLINK Linker, cliquer sur l'option « *Suppress COD-file generation* ». Cliquer sur *OK*.

3.3.4. Édition du programme

- Créer un nouveau fichier dans le projet, *File >> Add New Files to Project…*
- L'enregistrer dans le répertoire projet avec l'extension *c* (ex : *clignotant_C.c*)
- MPLAB Editor se lance
- Saisir le programme et enregistrer.

Penser à ajouter des commentaires au programme pour montrer que vous avez bien compris les différentes instructions.

3.3.5. Compilation

La compilation et l'édition de lien se fait en cliquant sur l'icône  de la barre de menu. Comme en assembleur, les erreurs et warnings sont signalés dans la fenêtre « *Output* » ainsi que le résultat de la compilation *BUILD SUCCEEDED* ou *BUILD FAILED.*

En double-cliquant sur une erreur ou un warning dans la fenêtre « Output », vous serez amené directement à la ligne concernée dans votre programme.

3.4. Simulation et programmation du micro-contrôleur

Les procédures de simulation et de transfert du code exécutable sur le micro-contrôleur ne diffèrent pas de celles que vous avez pu mettre en œuvre pour le développement en assembleur ; *cf*. l'énoncé du premier TP.

Effectuez la simulation du générateur de signal carré donné ci-avant dans l'énoncé.

3.5. Applications

3.5.1. Chenillard sans interruption

On souhaite créer un chenillard à motif variable. Un motif étant défini par une séquence d'éclairage des LED comme par exemple {0x01, 0x02, 0x04, 0x08}, cette séquence est répétée à l'infini.

Donner l'algorigramme du chenillard défini par la séquence de motif {0x01, 0x02, 0x04, 0x08}. Chaque motif devra rester allumé pendant un temps de 500 ms.

Pour définir la séquence de motifs, vous prendrez soin de déclarer un tableau de unsigned char car ce type correspond à un format 8 bits qui pourra directement être affecté sur le PORT B (cf., MPLAB C18 C COMPILER *USER'S GUIDE, page 11).*

Écrire et tester ce programme.

3.5.2. Chenillard multiple sans interruptions

On souhaite ajouter un second motif distinct au chenillard de manière à permettre le choix de la séquence. On prendra par exemple comme seconde séquence {0x01, 0x02, 0x04, 0x08, 0x04, 0x02}. La séquence considérée sera sélectionnée par une variable dans la fonction principale (*ie*., la fonction *main()*). Le motif à afficher à l'intérieur de la séquence considérée sera retourné par une fonction appelée *valeur_chenillard()* dont vous définirez les entrées et les sorties.

Donner l'algorigramme du chenillard défini par le couple de séquences {0x01, 0x02, 0x04, 0x08} et {0x01, 0x02, 0x04, 0x08, 0x04, 0x02}.

Écrire et tester ce programme.

4. TP n°2 : Interruptions et Périphériques

Ce deuxième TP est divisé en deux parties : des définitions concernant le module TIMER0, et une application.

4.1. Interruption de débordement du TIMER0

Nous devons maintenant faire clignoter la led L1, broche RB0, à une fréquence de 0.5Hz sans mobiliser le microcontrôleur. Il n'est donc plus question d'utiliser une temporisation pour compter le temps (cf. TP n°1). Nous allons demander à un module externe : le TIMER0, de compter le temps et de signaler la fin du comptage au microcontôleur par une interruption dite interruption de débordement. La figure 8 montre le signal de sortie du TIMER0 et les interruptions de débordement associées. Le TIMER0 compte jusqu'à la valeur *max* en *T* secondes. Toutes les *T* secondes, l'état de la led L1 est inversé.

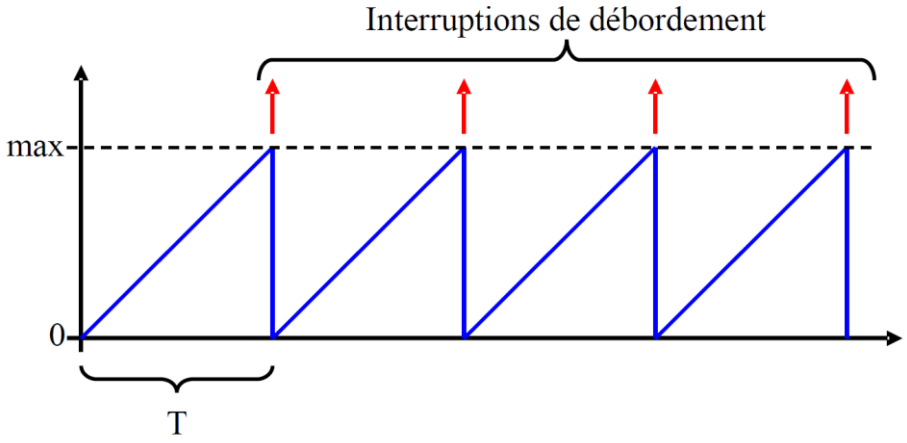

Figure 8: Interruption de débordement du TIMER0

Trouver, en utilisant la datasheet, l'expression de la valeur *max* et de la période de comptage *T* en fonction de la configuration du TIMER0. (*cf datasheet p123*)

Qu'elle est la configuration pour une période de comptage *T* d'environ 1s ?

Quels sont les bits à initialiser pour activer l'interruption du TIMER0 ?

Expliquer la configuration du registre T0CON.

L'énoncé demande de faire clignoter la led à une fréquence de 0.5Hz donc une période de comptage de 1s. Avec la configuration choisi précédemment, on a une interruption toutes les (1+*t*)s, si on souhaite mesuer le temps précisement cette erreur de *t* seconde n'est pas négligeable.

Pour compenser cette erreur décalage, l'idée est d'initialiser les registres TMR0H TMR0L afin que le comptage ne commence pas à 0 mais à une valeur *min* permettant d'annuler *t* comme le montre la figure 9.

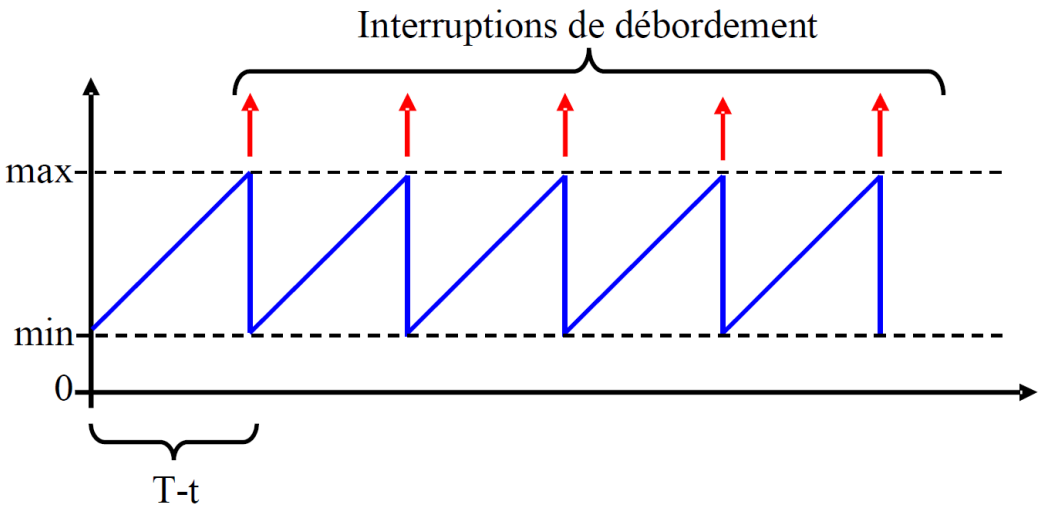

Figure 9: Interruption de débordement du TIMER0

Calculer la valeur *min* pour avoir une fréquence de comptage de 1s exactement.

4.2. Application : clignotant

Une des difficultés consiste alors à configurer correctement le TIMER0 de manière à permettre de générer les interruptions périodiquement. On donne ci-après les éléments du code source pour ce faire.

```
//-------------------------------------------------
//  Configuration  des  interruptions,  cf.  sections  9  et
// 11 du datasheet et le fichier header p18h4520.h
//-------------------------------------------------

      // active l'interruption pour le TIMER0
      INTCON = VALEUR;

      // Place l'interruption TIMER0 en priorite haute
      INTCON2 = VALEUR;

      // Active la gestion des priorites pour les interruptions
      RCONbits.IPEN =  VALEUR;

      // Efface le registre TMR0H
      TMR0H       = VALEUR;
      // Efface le registre TMR0L
      TMR0L       = VALEUR;

      // Active le TIMER0 et configure la valeur du prescaler
      T0CON       = VALEUR;

      // Active les interruptions
      INTCONbits.GIE = VALEUR;
```

*Configuration du micro-contrôleur pour la gestion d'interruption et configuration du TIMER0 pour (**à compléter**).*

L'objectif est alors de déclencher une interruption à chaque fois que le TIMER0 est en « overflow », cette interruption commandant le changement de motif du chenillard. A toutes fins utiles, on rappelle que le langage C ne permet pas directement de traiter les interruptions puisqu'il ne laisse pas à l'utilisateur le contrôle des adresses du

programme (équivalent du *org* en assembleur). A la place une directive *#pragma* a donc été intégrée au compilateur :

– Les directives

```
#pragma code  mon_prog = @
void mon_prog(void)
      {
      code C ...
      }
#pragma code
```

forcent le compilateur à placer le *code C* à l'adresse *@* (*cf.*, MPLAB C18 C COMPILER USER'S GUIDE, page 20 et 29).

– La directive

```
#pragma interrupt ma_fonction
```

déclare *ma_fonction* comme la fonction de traitement des interruptions prioritaires (adresse 0x18).

– La directive

```
#pragma interruptlow ma_fonction
```

déclare *ma_fonction* comme la fonction de traitement des interruptions non prioritaires (adresse 0x08).

Rappel : un exemple de programme C avec interruption est disponible dans le cours.

Donner les algorigrammes et fonctions C du programme principal et du programme d'interruption pour que la led L1 clignote toute les secondes. Écrire et tester ce programme.