Aix*Marseille université

Faculté des Sciences

Licence 3 SPI

Informatique industrielle

Travaux Dirigés

Julien Marot

Aix*Marseille université

Faculté des Sciences

Licence 3 SPI

Informatique industrielle

Travaux Dirigés

Les interruptions
Gestion de l'espace mémoire

Julien Marot

Les figures sont utiles aux deux exercices indistinctement.

1 Exercice 1 : taille de l'espace mémoire

Déterminez d'après la figure 1 la taille des mémoires programmes et données.
1) Comparez les résultats obtenus aux information données à la figure 2.
Retrouvez-vous les 32kbytes annoncés ?
2) Idem pour la figure 3. Retrouvez-vous les 3.9 kbytes annoncés ?

2 Exercice 2 : gestion de l'espace mémoire et des interruptions

Notez l'existence de cases mémoire spécifiques dans la figure 2 : RESET, et Interruptions haute et basse.

Le vecteur RESET est un pointeur vers l'adresse mémoire du début du programme principal : il fournit au Program Counter l'adresse de la première instruction du programme.
Le vecteur interruption (basse ou haute) pointe vers l'adresse mémoire du programme à exécuter en cas d'interruption.

En gardant en tête ces définitions, répondez aux questions suivantes :

1. Numérotez les étapes du programme d'interruption dans l'algorigramme de la figure 4 ;
2. Repérez ces étapes dans le code en section 3 ;
3. Quelle est l'unique case mémoire de la figure 2 qui est explicitement dédiée au programme principal ?
4. Quelles sont les deux seules cases mémoire de la figure 2 qui sont explicitement dédiées à un programme d'interruption ?
5. D'après le programme présenté en section 3, quelle est la ligne de code qui est contenue par la case mémoire dont l'adresse est '0000' ? Par celle dont l'adresse est '0008' ?
6. Ces lignes de code permettent de se placer à un emplacement particulier de l'espace mémoire. Déduisez-en la nature du label 'init'. Faites un schéma sur lequel apparaîtront la case mémoire d'adresse '0000', le program counter, et les cases mémoires contenant les deux premières instructions.
7. Quelles sont les cases mémoires de la figure 2 qui sont interdites à l'écriture du programme principal ? Dans quel cas, où le programme principal est mal placé dans la mémoire, une ligne de ce programme serait-elle écrasée et remplacée par `goto routine_int` ?
8. Dans l'exemple donné en section 3, l'instruction 'org ...' est suivie d'une instruction dont l'opcode est `goto`. Quel autre opcode aurait-on pu trouver ?

9. Repérez par une accolade une partie de la mémoire programme (figure 2) qui pourrait contenir le programme principal.
10. Admettons que le label 'init' code l'adresse `0x0180`. D'après le programme en section 3, que contient la case mémoire d'adresse `0x0180` après lecture du vecteur RESET ?
11. Repérez par une autre accolade la partie de la mémoire programme qui peut contenir le programme d'interruption.

3 Programme interruption : extraits

```
org h'0000';Adresse de début du programme sur Reset
goto init

org h'0008';Adresse de début du programme d'interruption
goto routine_int

init clrf  PORTB ;Remise à zéro des bascules D du port B
movlw h'00'
movwf TRISB ;le port B est défini en sortie
movlw h'83'
movwf T0CON ;TIMER0 On, 16bits, Prescaler 16
rcall tmr0_init ;Init TMR0 pour 1s pile
movlw h'A0'
movwf INTCON ;Autorisation Générale des IT et TMR0 IT

boucle nop ;Ne rien faire
goto boucle ;Boucle infini

tmr0_init movlw h'FF';Init du registre TMR0
movwf  TMR0H ;pour avoir 1 seconde pile
movlw h'FD'
movwf  TMR0L
return

;----- Routine d'interruption
routine_int
movwf  W_TEMP  ;Sauvegarde de W
movff  STATUS, STATUS_TEMP  ;Sauvegarde de STATUS
movff  BSR, BSR_TEMP  ; Sauvegarde de BSR
btfsc  INTCON,2  ;TMR0IF == 1 ?
rcall  tmr0_overflow ;traitement de l'IT TMR0IF
movff BSR_TEMP, BSR  ;Restauration de BSR
movff  W_TEMP, WREG  ;Restauration de W
movff STATUS_TEMP, STATUS ;Restauration de STATUS
retfie  ;Retour au programme principal

;----- Interruption de débordement du TIMER0
tmr0_overflow
bcf  INTCON,2  ;Suppression du flag d'interruption
rcall tmr0_init ;Init TMR0 pour 1s pile
movlw  h'01'
xorwf  PORTB  ;Inversion de l'état de la LED l1, RB0
return
```

4 Figures

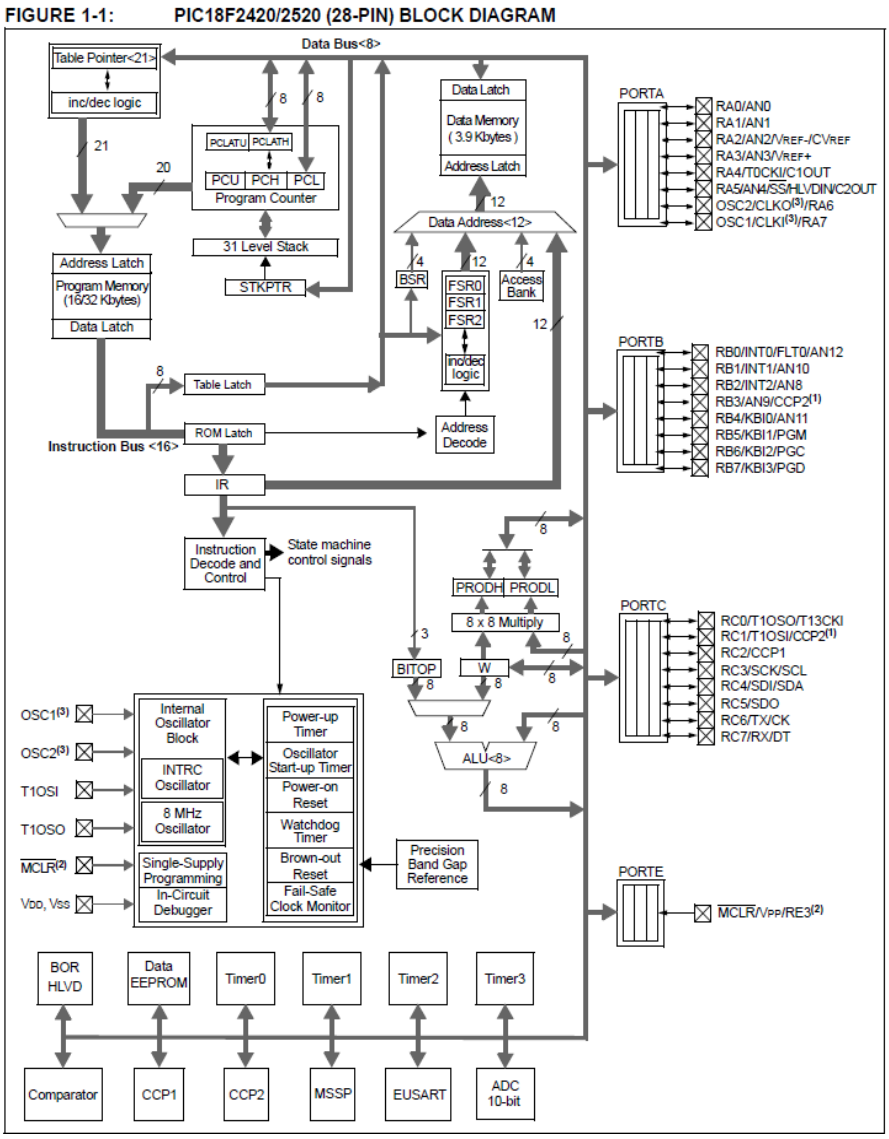

FIGURE 1 – Schéma de principe du PIC

FIGURE 5-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F2420/2520/4420/4520 DEVICES

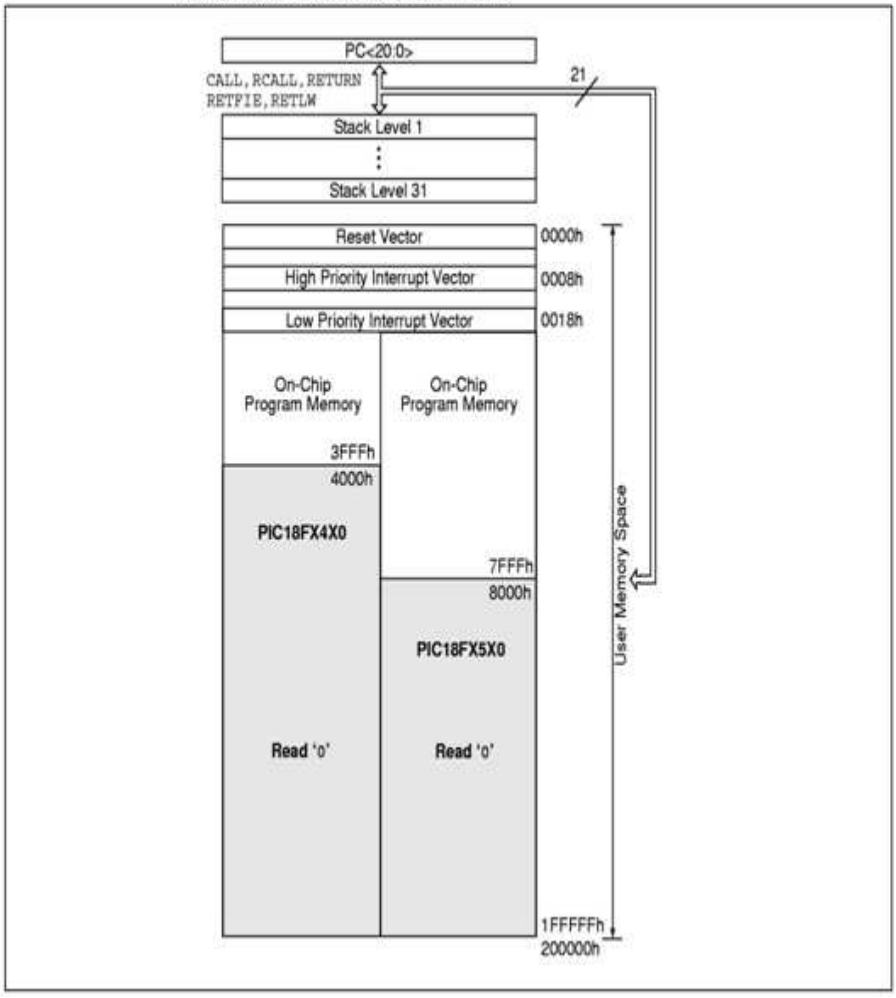

FIGURE 2 – Mémoire programme

FIGURE 5-6: DATA MEMORY MAP FOR PIC18F2520/4520 DEVICES

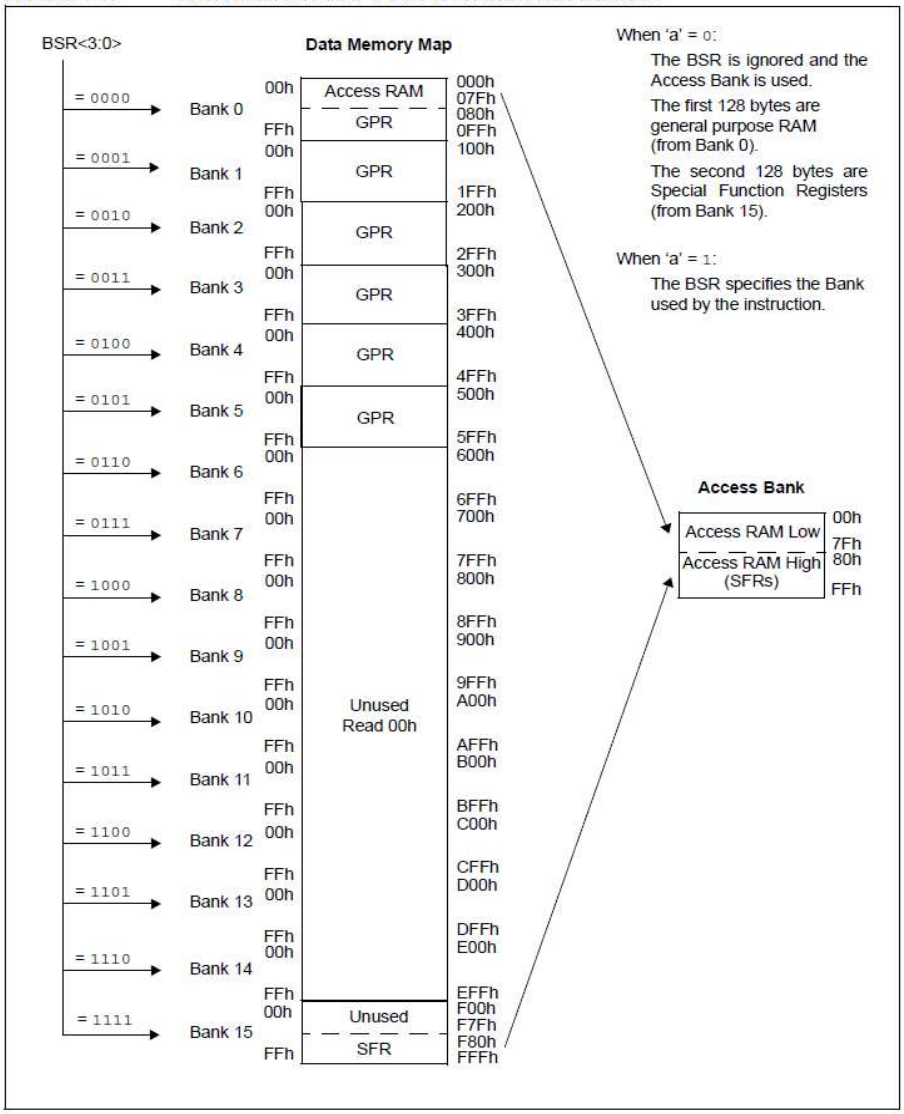

FIGURE 3 – Mémoire données

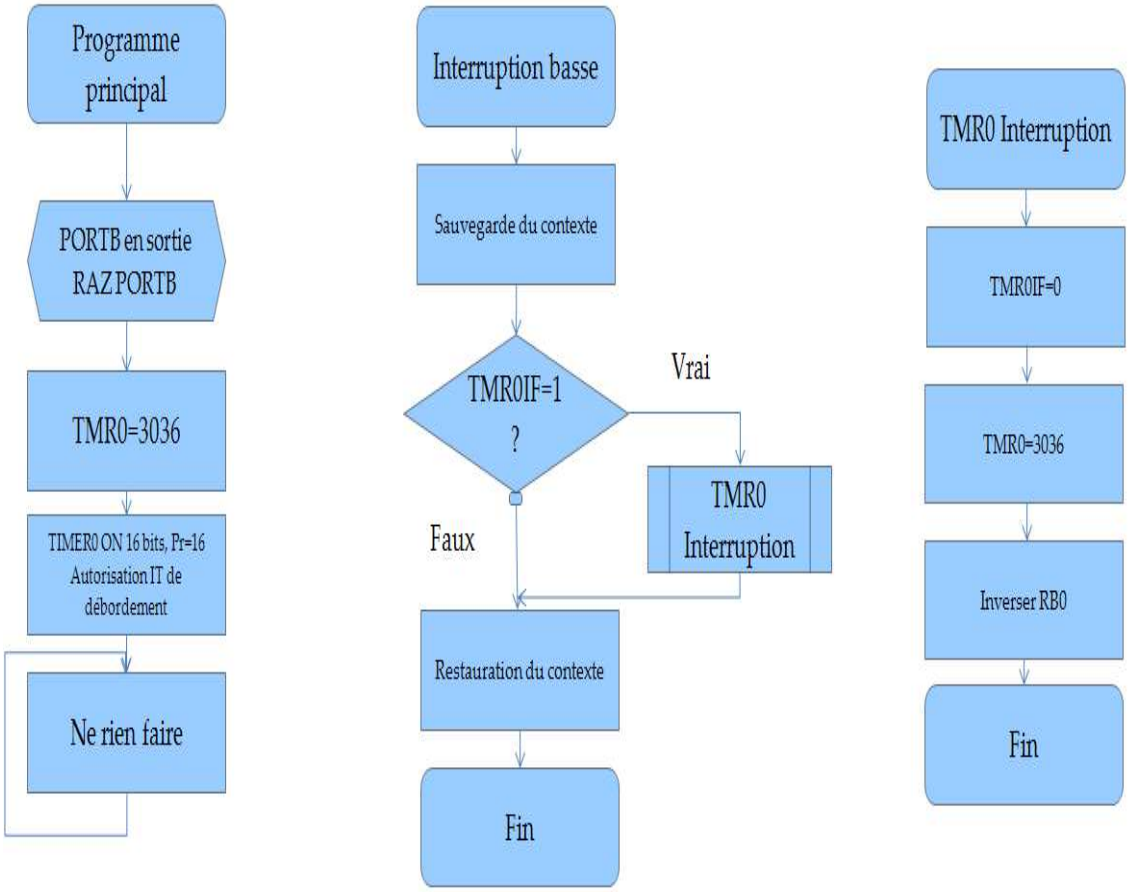

FIGURE 4 – Algorigrammes

Aix*Marseille université

Faculté des Sciences

Licence 3 SPI

Informatique industrielle

Travaux Dirigés

Pointeurs
pour fonctions en langage C

Julien Marot

Ce TD porte sur la gestion des entrées et sorties dans des programmes en C.

1 Introduction

Le schéma des fonctions en C n'inclut que des entrées, pas des sorties. La problématique de ce TD est donc la suivante.

1.1 Problématique

Il est nécessaire de trouver un moyen de modifier une valeur tout en rendant cette modification visible en dehors de la fonction considérée. Lors de ce TD, vous travaillerez sous Linux avec un compilateur nommé `gcc`, qui est installé par défaut sur la distribution Linux de votre salle de TP (il faut donc lancer les ordinateurs sous Linux). Premièrement, vous créerez un modèle de programme en C qui inclut trois fichiers, pour faire tourner une fonction. Vous remarquerez qu'il n'existe qu'une sortie à une fonction C. C'est pourquoi, dans un deuxième temps, vous étudierez l'intérêt du passage par adresse pour distinguer les entrées des sorties. Ensuite, vous trouverez un contre-exemple à la méthode que vous aurez créée pour gérer les entrées/sorties en C.

1.2 Pointeurs et espace mémoire

Afin de modifier une valeur dans une fonction et de rendre 'visible' cette modification hors de la fonction, on entre l'adresse de la valeur à modifier, afin de placer à cette adresse le résultat du calcul à effectuer par la fonction. Dans ce TD, on souhaite montrer l'intérêt du passage par adresse.

Question 1 : Comment un pointeur permet-il d'accéder à l'espace mémoire ?

2 Programmes C sous Linux

Dans cette section, on vous donne un modèle pour les trois fichiers qui constituent un programme en C. Lisez cette section en entier avant de répondre aux questions.

2.1 Modèle de programme en C

Lorsqu'on programme en langage C, il est conseillé de faire trois fichiers : un fichier `main.c`, un fichier `All_Functions.h`, et un fichier `All_Functions.c`.

Le programme `main.c` prend la forme suivante :

```
#include <stdio.h>
#include <math.h>
#include <dirent.h>

#include "All_Functions.h"
#include "All_Functions.c"

int main(void)
    {
printf("\nPlease enter entree1 :\n");
scanf("%f",&entree1);

printf("\n\n entree1 = %f\n",entree1);

//Appel de fonction

return 0;

}
```

Le programme `All_Functions.c` prend la forme suivante :

```
#include "All_Functions.h"

static const double pi = 3.14159265358979323846;

// Déclaration et initialisation des sorties du programme
    float entree1=0;

void Fonction1 (float entree1)
{
// Opérations
}
```

Vous remarquerez que le fichier `All_Functions.c` appelle un fichier `All_Functions.h`.

Le programme `All_Functions.h` prend la forme suivante :

```
#ifndef All_Functions_h
#define All_Functions_h

void Fonction1 (float entree1);

#endif
```

Question 2 : Lancez les ordinateurs sous Linux. Vous créerez un dossier dans, et à l'intérieur trois 'documents vides'.

Question 3 : Avec l'éditeur `gedit` créez les fichiers `main.c`, `All_Functions.c` et `All_Functions.h`.

Vous remarquerez que la fonction C créée ci-dessus ne retourne rien (`void`) ; Une fonction C retourne en général une seule sortie qui peut être de type entier, float, etc... indiqué dans la déclaration et la définition de la fonction. Ainsi, il est impossible de retourner directement plusieurs sorties.

2.2 Compilation et exécution sous Linux

La compilation consiste à créer un fichier exécutable à partir d'un programme en C (le code source).

Question 4 : Au préalable, placez-vous dans le même répertoire que le fichier source. Pour cela, utilisez la commande `cd`.

Question 5 : Compilez le code source main.c pour produire une sortie du nom de main.exe. La compilation se fait *via* un simple terminal en tapant à la ligne de commande :

```
gcc main.c  -o main.exe
```

Question 6 : Exécutez le binaire qui vient d'être compilé. Pour cela, il suffit ensuite de taper dans le même terminal (et toujours en étant toujours dans le même répertoire) :

```
./main.exe
```

3 Intérêt du passage par adresse

3.1 Distinction des entrées et des sorties

Avec l'éditeur `gedit` modifiez les fichiers de façon à ce que la Fonction1 prenne plusieurs entrées. Certaines seront des valeurs, et d'autres seront des pointeurs.

3.1.1 Fichier de déclaration .h

Modifiez le fichier `All_Functions.h` qui comprendra la déclaration des fonctions.

```
void Fonction1 (
                          float entree1, float entree2,
                           float* pointeursortie1, float* pointeursortie2);
```

Les entrées sont des valeurs de type float, les sorties sont des pointeurs de type float.

3.1.2 Fichier de définition .c

Modifiez le fichier `All_Functions.c` qui comprendra la définition des fonctions.

```
void Fonction1 (float entree1, float entree2,
                float* pointeursortie1, float* pointeursortie2)
{
        *pointeursortie1 = ??;
        *pointeursortie2= ??;
       }
```

Les entrées sont des valeurs de type float, les sorties sont des pointeurs de type float, et dans la fonction on modifie la valeur pointée par chaque pointeur.

3.1.3 Fichier principal

Vous compléterez le fichier `main.c` pour qu'il appelle la fonction écrite précédemment.
Question 7 : Programmez l'affichage des entrées et des sorties avant et après l'appel de la fonction. Quelles sont les valeurs qui sont modifiées ? Pourquoi ?

3.2 Contre-exemple

Vous créerez une fonction `Fonction2` dans laquelle vous créerez un contre-exemple : vous tenterez de renvoyer une sortie `sortievaleur` en passant sa valeur dans la fonction et en la modifiant dans la fonction.

Question 8 : Que remarquez-vous quant à la valeur des sorties des fonctions ? En particulier, `sortievaleur` est-elle modifiée par rapport à sa valeur d'initialisation, quand on l'affiche après l'appel de la fonction ? Pourquoi ?

Aix*Marseille université

Faculté des Sciences

Licence 3 SPI

Informatique industrielle

Travaux Dirigés

Afficheur LCD

Langage C

Julien Marot

Ce TD porte sur la création de fonctions en langage C pour l'afficheur LCD de la maquette de test PICDEM2PLUS. Dans un premier temps, vous transcrirez les fonctions de base du langage assembleur en C. Puis vous écrirez le code en C que vous utiliserez en TP pour piloter l'écran LCD de la maquette.

Pour tout ce TD, vous pourrez utiliser des opérateurs décrits en figure 5, en annexe 4.

1 Exercice 1 : Introduction

Dans un premier temps on s'intéresse aux équivalences entre langage assembleur et langage C. Considérez le code assembleur suivant (il s'agit de la configuration du PORTB) :

```
movlw 0x00
movwf TRISB

movlw 0x00
movwf PORTB

movlw 0x0F
movwf ADCON1
```

Considérez maintenant le code qui inverse l'état de la broche 1 du PORTB :

```
movlw 0x01
xorwf PORTB
```

Question : quel est pour chaque programme l'équivalent en C du code assembleur ci-dessus ?

2 Exercice 2 : pilotage de l'écran LCD

L'exercice 2 consiste à transcrire le code assembleur qui permet de piloter l'écran LCD en code C. La sous-section 2.1 concerne le transfert d'information telles des chiffres et des lettres vers l'écran. La sous-section 2.2 concerne l'envoi d'ordres vers l'écran, afin par exemple de l'effacer.

La figure 6 en annexe vous donne le schéma de câblage de l'afficheur, avec les bits du PORTD qui sont impliqués dans son fonctionnement.

Dans chaque sous-section, vous devrez compléter un code en C, dans la partie "TRAVAIL A FAIRE".

2.1 Transfert de d'information

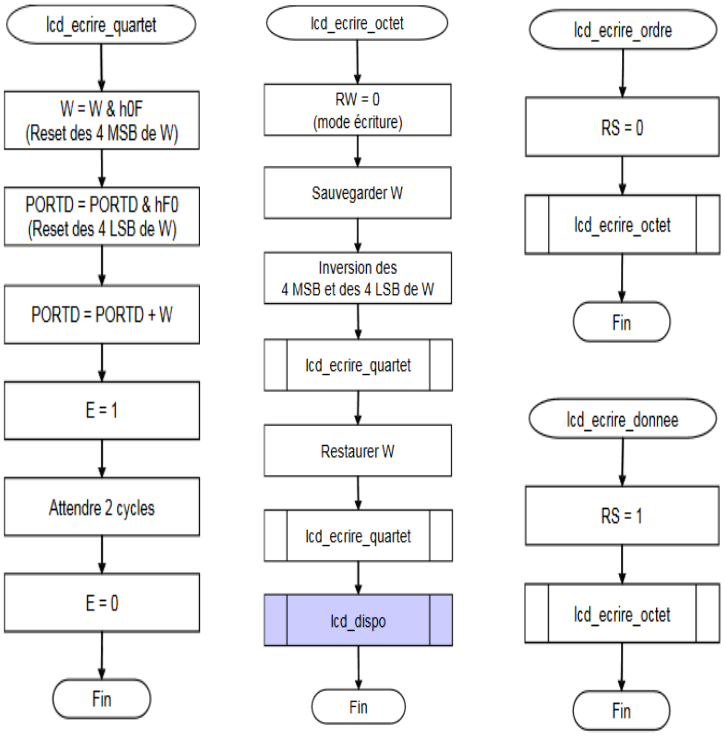

FIGURE 1 – Algorigrammes

```
lcd_ecrire_quartet
    andlw   h'0F'       ;Mise à 0 des 4 MSB de W W=0000XXXX
    bcf PORTD,3     ;Reset de RD3...RD0 pour effectuer un masquage
    bcf PORTD,2
    bcf PORTD,1
    bcf PORTD,0
    iorwf   PORTD       ;Mise à jour du bus DATA (RD0...RD3)
    bsf     PORTA,1     ;La ligne E passe à 1
    nop         ;Attente de 2 cycles d'horloge
    nop
    bcf     PORTA,1     ;La ligne E passe à 0
    return          ;Retour

lcd_ecrire_octect
    movwf   lcd_db
    bcf PORTA,2     ;La ligne RW est à 0 mode ecriture
    swapf   lcd_db,0        ;Inversion des 4 MSB et LSB de lcd_db, le résultat est placé dans W
    rcall   lcd_ecrire_quartet  ;Appel du sous programme d'envoi pour le quartet MSB
    movf    lcd_db,0        ;Chargement de lcd_db dans W
    rcall   lcd_ecrire_quartet  ;Appel du sous programme d'envoi pour le quartet LSB
    rcall   lcd_dispo
    return

lcd_ecrire_ordre
    bcf PORTA,3     ;La ligne RS est à 0 mode ordre
    rcall   lcd_ecrire_octect
    return

lcd_ecrire_donnee
    bsf PORTA,3     ;La ligne RS est à 1 mode données
    rcall   lcd_ecrire_octect
    return
```

FIGURE 2 – Programmes assembleur

TRAVAIL A FAIRE : Programmes en C à compléter

Directives au préprocesseur et définitions

```
#include <delays.h>
#include <p18f4520.h>
#pragma config WDT = OFF

// Definition : bus de controle de l'afficheur -- version carte "verte" PICDEM2+ ROHS
#define Enable  PORTDbits.RD6
#define RW  PORTDbits.RD5
#define RS  PORTDbits.RD4
#define LCD_ON_OFF PORTDbits.RD7

// Definition : masques pour affectation uniquement du bus data du LCD placé en RD0:RD3
#define masque_LSB  0x0F
#define masque_MSB  0xF0
```

QUARTET

```
void lcd_ecrire_quartet (unsigned char quartet_dans_octet)
{
// Envoi du quartet sur le port D <=> bus de donnee du contrôleur d'affichage
quartet_dans_octet  &=     _   _   _    ;  // 1 - mise à 0 du MSB de quartet_dans_octet
PORTD      &=     _   _   _    ;  // 2 - mise à 0 du LSB du portD
PORTD  |= quartet_dans_octet;  // 3 - Affectation du LSB de quartet_dans_octet
//Ou bit à bit puis affectation

// Generation de la sequence de synchronisation avec le controleur d'affichage
    _   _   _      = 1 ;  // E passe à 1...
Delay1TCY(); // Attente de deux cycles...
Delay1TCY();
    _   _   _      = 0 ;  // E passe à 0...
}
```

OCTET

```
void lcd_ecrire_octet (unsigned char octet)
{
unsigned char dummy;

// swap MSB / LSB pour envoyer d'abord le MSB
octet  = (octet << 4 | octet >> 4) ;

// Envoi du MSB...
lcd_ecrire_quartet(octet);

// swap LSB / MSB pour envoyer le LSB
octet  =     _   _   _    ;

// Envoi du LSB...
    _   _   _        (       _   _   _        );

// Test de disponibilite du controleur d'affichage...
dummy  = lcd_dispo();
}
```

ORDRE

```
void lcd_ecrire_ordre (unsigned char octet)
{
// ecriture sur le bus data du LCD
// l'octet transmis doit etre interprete comme un ordre
    _   _   _       =        _   _   _        ;

    _   _   _       =        _   _   _        ;


// Envoi du MSB...
lcd_ecrire_octet(octet);
}
```

DONNEE

```
void lcd_ecrire_donnee (unsigned char octet)
{
// ecriture sur le bus data du LCD
// l'octet transmis doit eter interprete comme une donnee
    _   _   _       =        _   _   _        ;

    _   _   _       =        _   _   _        ;

// Envoi du MSB...
lcd_ecrire_octet(octet);
}
```

2.2 Effacement d'écran et changement de ligne

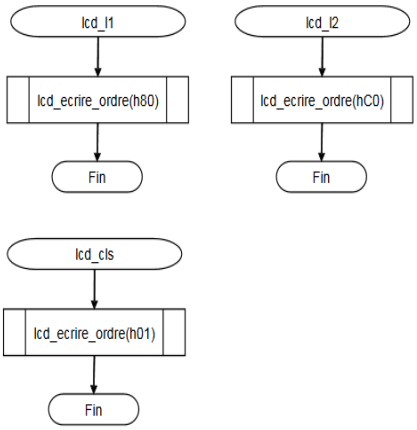

FIGURE 3 –

```
lcd_l1
    movlw   h'80'
    rcall   lcd_ecrire_ordre
    return

lcd_l2
    movlw   h'C0'
    rcall   lcd_ecrire_ordre
    return

lcd_cls
    movlw   h'01'
    rcall   lcd_ecrire_ordre
    return
```

FIGURE 4 –

TRAVAIL A FAIRE : Programmes en C à compléter

Voici les fonctions à compléter :

L1

```
void lcd_l1 (void)
{
    _   _   _       (       _   _   _       ); // ordre 1000 0000 en hexa
}
```

L2

```
void lcd_l2 (void)
{
    _   _   _       (       _   _   _       ); // ordre 1100 0000 en hexa
}
```

CLS

```
void lcd_cls (void)
{
    _   _   _       (       _   _   _       );   // ordre 0000 0001 en hexa
}
```

3 Exercice 3 : Création des programmes

Lancez votre ordinateur sous Linux, et créez un nouveau fichier. Nommez-le lcd.c, et ouvrez-le avec 'gedit'.

Travail à faire : Ecrivez chaque fonction complétée ci-dessus dans le fichier lcd.c.

Travail à faire : Une fois le fichier lcd.c rédigé, enregistrez-le dans 'Mes Documents'. Ce fichier vous sera utile lors des TPs.

4 Annexe

Classe d'opérateur	Opérateur(s)	Associativité
Parenthèsage	()	de gauche à droite
Suffixes	[] -> . ++ —	de gauche à droite
Unaires	& * + - ! sizeof ~	de droite à gauche
Changement de type	(type)	de droite à gauche
Multiplicatifs	* / %	de gauche à droite
Additifs	+ -	de gauche à droite
Décalage	<< >>	de gauche à droite
Comparaisons	< <= > >=	de gauche à droite
Égalités	== !=	de gauche à droite
ET bit à bit	&	de gauche à droite
OU exclusif bit à bit	^	de gauche à droite
OU bit à bit	|	de gauche à droite
ET logique	&&	de gauche à droite
OU logique	||	de gauche à droite
Condition	? :	de droite à gauche
Affectations	= += -= *= /= &= |= ^= <<= >>=	de gauche à droite
Succession	,	de gauche à droite

FIGURE 5 – Opérateurs en C

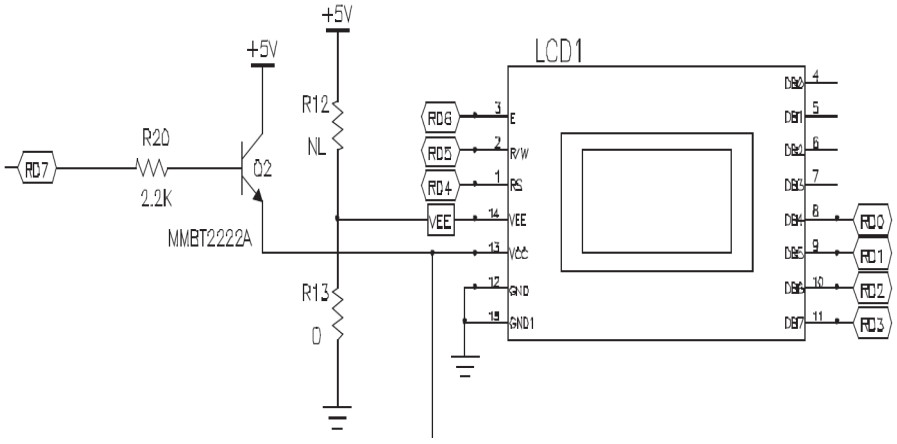

FIGURE 6 – Schéma de câblage de l'afficheur