Universite Saint Joseph De Beyrouth

Ecole supérieure des ingénieurs de Beyrouth (ESIB)

**Système à microprocesseur**

# Projet jeu Mastermind

**Présenté par :**

**GHAFARI Lea (191184)**

**IBRAHIM Mohamad (191545)**

**Présenté a :**

**Dr. André Chkaibane**

# Table de matière

# Développement du jeu Mastermind

# en langage assembleur

Ce rapport décrit la réalisation du jeu Mastermind utilisant le langage en assembleur, le logiciel MPLab et Proteus ISIS

La connaissance du travail dans l'outil logiciel est démontrée ainsi que la connaissance avancée de l'assembleur. Le produit final devrait permettre à l'utilisateur de jouer le jeu et laisser à son adversaire de deviner la combinaison choisie.

Une méthodologie agile a été suivie tout au long du projet, afin d'en implémenter correctement les principales parties.

Le résultat final était un jeu entièrement fonctionnel qui utilise MPLab comme logiciel de base capable d'atteindre les principaux objectifs avec l'aide de Proteus

Le rapport suivant permettra aux lecteurs de mieux comprendre les différents outils, codes et méthodes utilises afin d'atteindre le produit final.

## 1-Introduction

Le rapport suivant fournit un algorithme à la réalisation du jeu populaire Mastermind.

L'idée était basique, le but était de développer le jeu mastermind appliquant le logiciel MPLab en assembleur.

Le langage en assembleur est un langage de programmation de bas niveau et est spécifique à une architecture de processeur (respectivement microprocesseur) particulière.

Etant l'un des langages de programmation les plus anciens, il ressemble le plus à un langage machine natif. Il fournit un accès direct au matériel informatique de l'ordinateur nécessitant de bien comprendre son architecture et son système d'exploitation.

Après plusieurs exemples faits en classe, nous sommes amenés à une conclusion que lors de la conception de microcontrôleurs, il est nécessaire de connaître le matériel sur lequel le logiciel sera exécuté pour écrire un algorithme optimal.

On peut conclure que l'assembleur est utilisé quand on a besoin d'un traitement rapide et efficace du signal.

Par la suite, une description du jeu sera exploitée utilisant l'architecture du pic18f4520.

## 2-Description schématique

Cette partie vise à montrer les tests et les circuits effectués sur Proteus Isis et puis les tests faits au laboratoire.

Premièrement on dispose d'un microcontrôleur pic18f4520 connecte à un 4 boutons poussoirs sur les ports RB change utilises par le joueur qui aident à insérer la combinaison et un bouton poussoir sur le port RB0 pour que l'utilisateur insère le code à deviner

Un écran LCD connecte au portC et aux ports RD0, RD1, RD2 pour aider à visualiser les chiffres insérer et savoir si le joueur a gagné ou perdu.

Finalement le bouton RESET est place et a pour rôle de réinitialiser le jeu quand le joueur le désire.

https://www.microchip.com/PIC18F4520

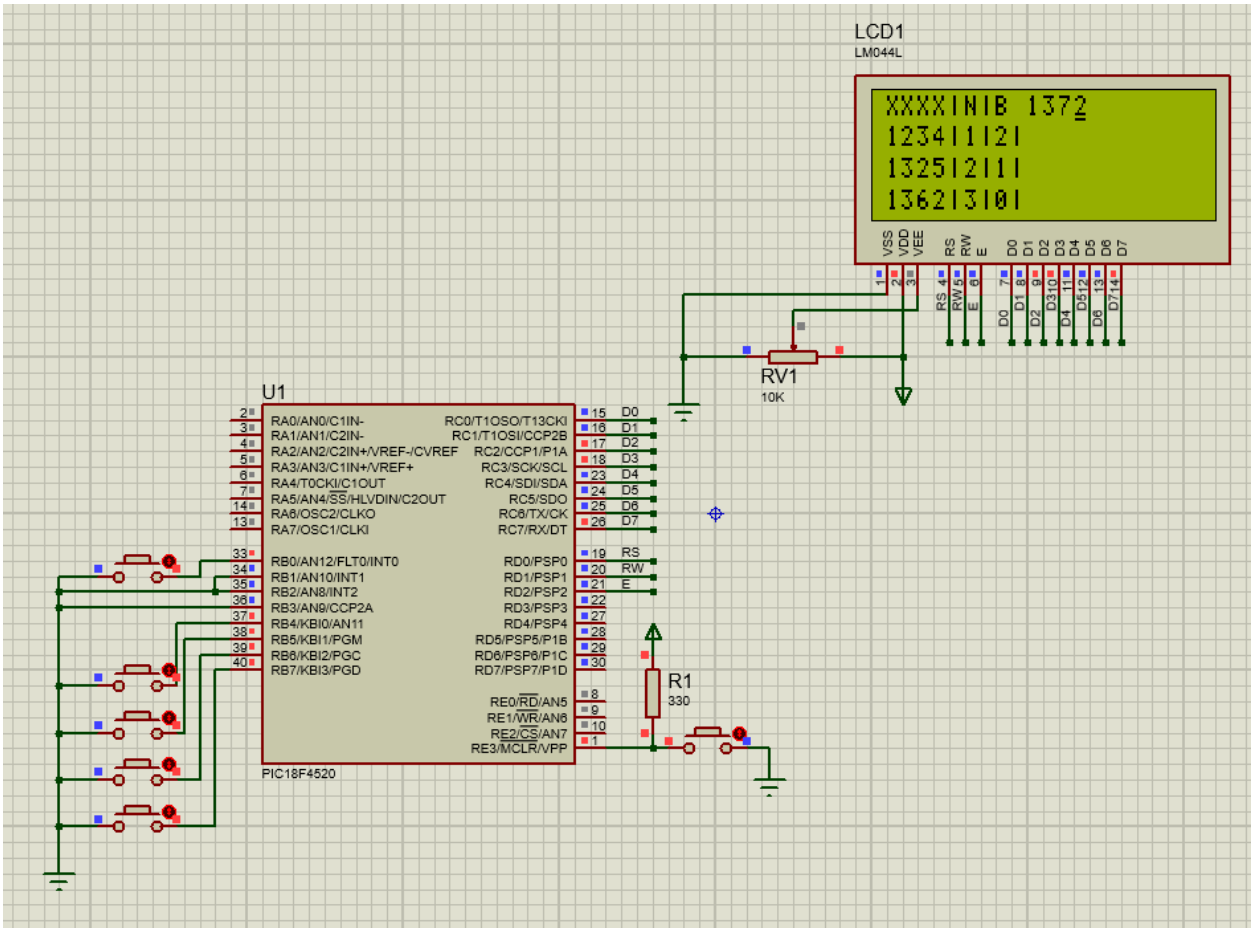



*Figure 2.1* circuit ISIS

## 3-Description de la réalisation du jeu

Le jeu est réalisé en utilisant la combinaison de chiffre déjà connue qui est à deviner.

Au début du jeu, le joueur essaie de deviner le terme en écrivant 4 chiffres allant de 1 vers 9 à partir de l'entrée standard. Si le joueur écrit une combinaison de chiffre, le nombre de chiffres correspondants par leurs valeurs uniquement sera indiqué par une variable B (blanc) affichée à l'écran LCD à côté de la variable N(Noir) qui est le nombre de chiffres correspondants par leurs valeurs et leurs positions aux chiffres cachés.

Si le joueur insère une combinaison de chiffres qui ne varie pas entre 1 et 9, comme ‘0000’, un message sur l’écran LCD apparait et lui demande de réinsérer un code.

Chaque chiffre est imprimé sur l’écran LCD, après chaque combinaison de 4 chiffres, le décodeur revient à la ligne pour une nouvelle combinaison, si le joueur ne la pas encore devine et que toutes ses possibilités de deviner se sont écroulées, l’écran LCD affiche qu’il a perdu et si au bout d’un certain nombre de coups à placer les 4 chiffres correspondent exactement à leurs valeurs et leurs positions à ceux du code caché, la manche est terminée.

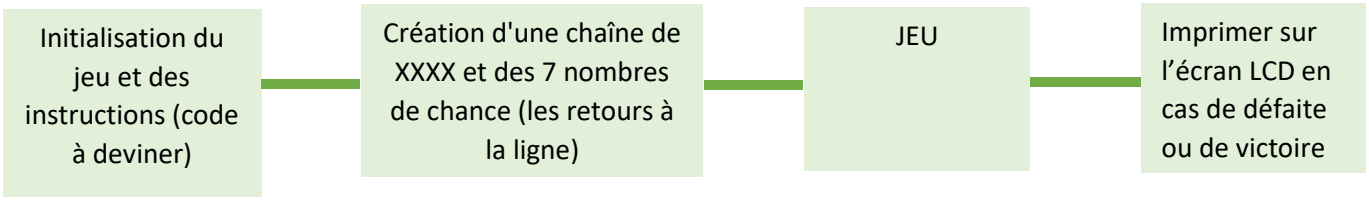



*Figure 3.1* diagramme simplifié du jeu

## 3.1 Initialisation des instructions

Au tout début du programme, nous imprimons les règles sous la forme d'une série de chaînes écrites dans la mémoire de programme du microcontrôleur

```
org 0x920
db "inserer un code:"  ;; sauvegarde des phrases dans la memoire ROM
```

*Figure 3.1.1* Chaine écrite dans la mémoire de programme

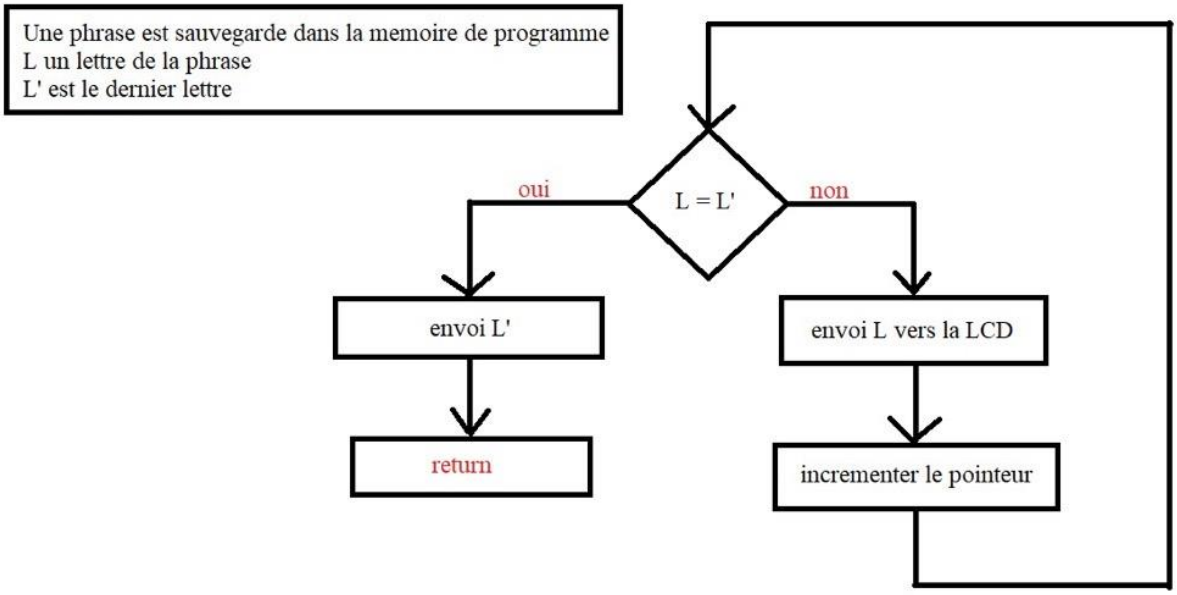
Pour commencer le jeu on a fait appel à la fonction display_inserer, et ses fonctions sœurs. Elle imprime le nombre que l’utilisateur doit faire deviner sur l’écran LCD à

partir de la position courante du pointeur initialisé dans init (tblptrh/tblptrl) jusqu'à ce qu'il atteigne le dernier caractère ‘ :’

```
ecrire_lcd_inserer
      tblrd*+
      movf tablat,w
      call DATAWRT
      call delay
      movlw A':'
      cpfseq tablat
      bra ecrire_lcd_inserer
```

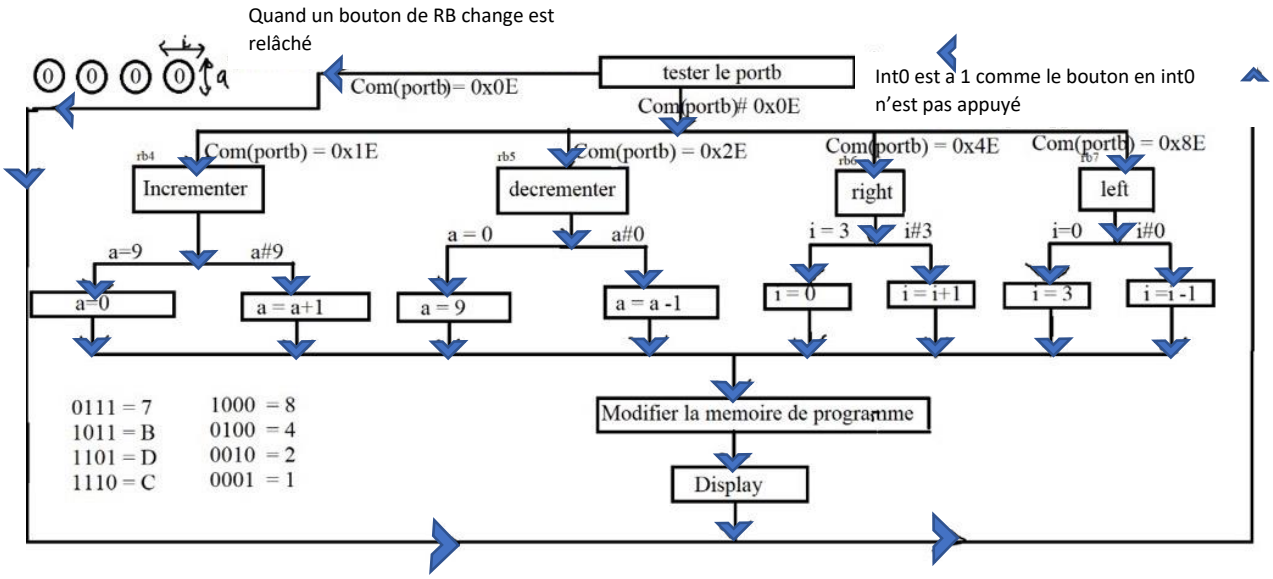
*Exemple d’affichage sur l’écran*

L’algorithme pour écrire n’importe quelle phrase est représentée par l’organigramme suivant :



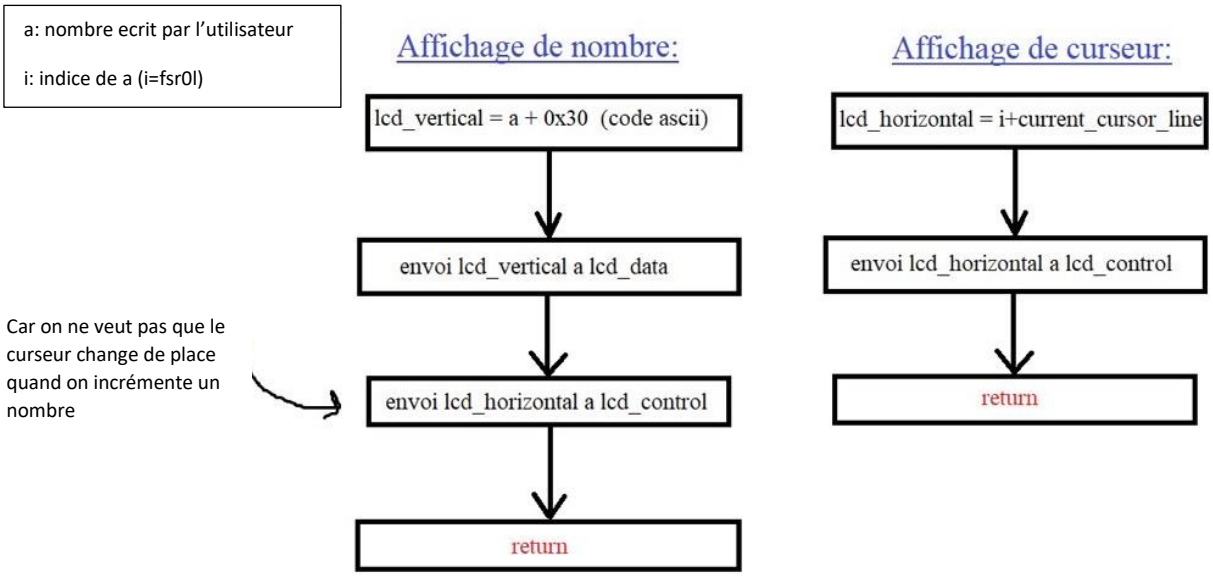
## 3.2 – Insérer le code à écrire

La première écriture du code par le joueur 1 (l'utilisateur) et le joueur 2 sera la même. Ils ne pourront écrire le code que s'il est dans l'interruption RB change, ils peuvent incrémenter et décrémenter à l'aide des curseurs et déplacer les curseurs a gauche et à droite (les boutons poussoirs RB4-> RB7) qu'on voit sur le schéma ISIS figure 2.1)




a: nombre ecrit par l'utilisateur

i: indice de a (i=fsr0l)




Car on ne veut pas que le curseur change de place quand on incrémente un nombre

Dans cet organigramme la variable current_cursor_line représente la première colonne de la ligne ou on est en train d'écrire le code.

## 3.3- Démarrage du jeu

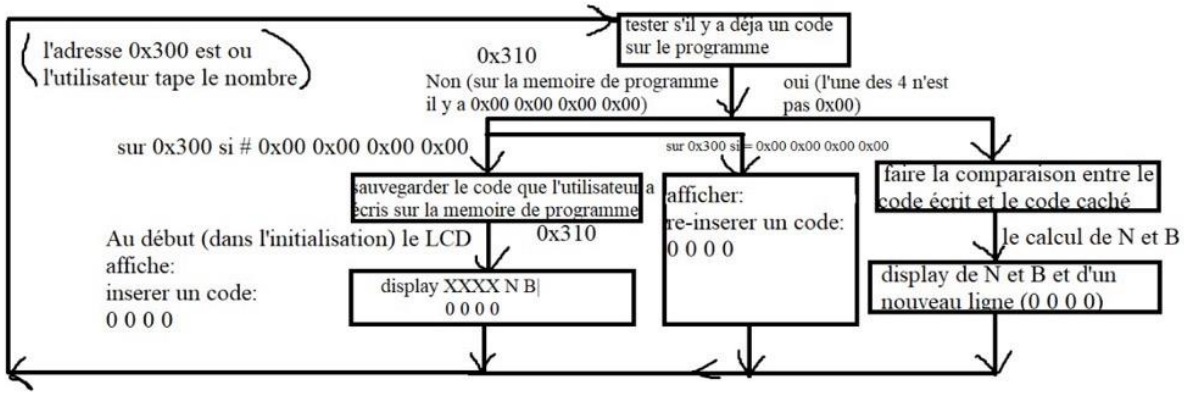
## Rôle du bouton enter

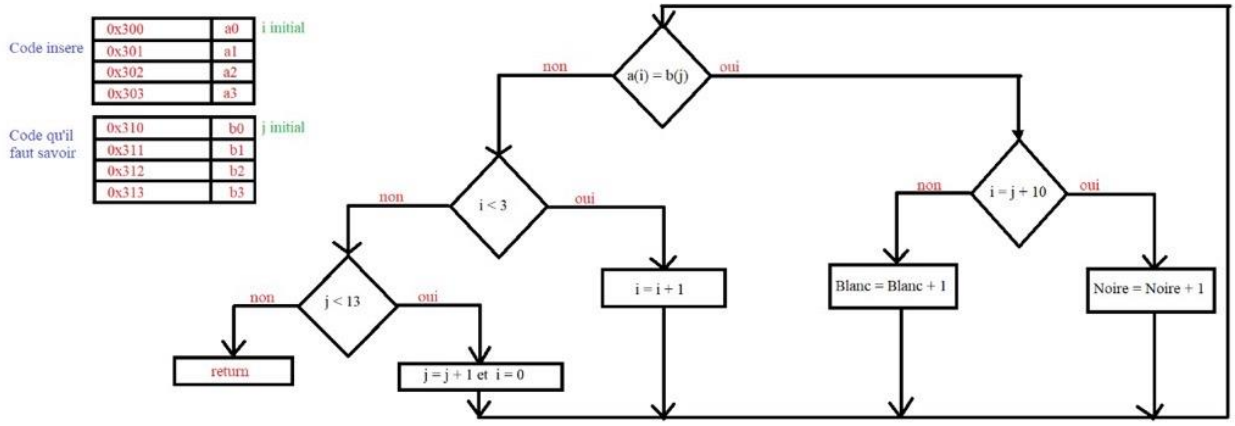
Le démarrage du jeu ne se fait qu'à travers int0

<u>Int0 a 3 rôles différents :</u>

1er est de démarrer le jeu et d'afficher XXXX, un 2e rôle est d'appuyer 'enter' pour que le joueur 2 puisse deviner donc son rôle est de comparer et finalement le 3e rôle est que si le joueur n'a rien écrit pour faire deviner, il y aura une demande de réinsérer un code

l'utilisateur va écrire un code formé de 4 chiffre, quand il decide de l'inserer (tester si ce code est vrai OU l'inserer comme le code qu'on doit savoir) il appuie sur le boutton (ENTER) (qui est en realité int0)

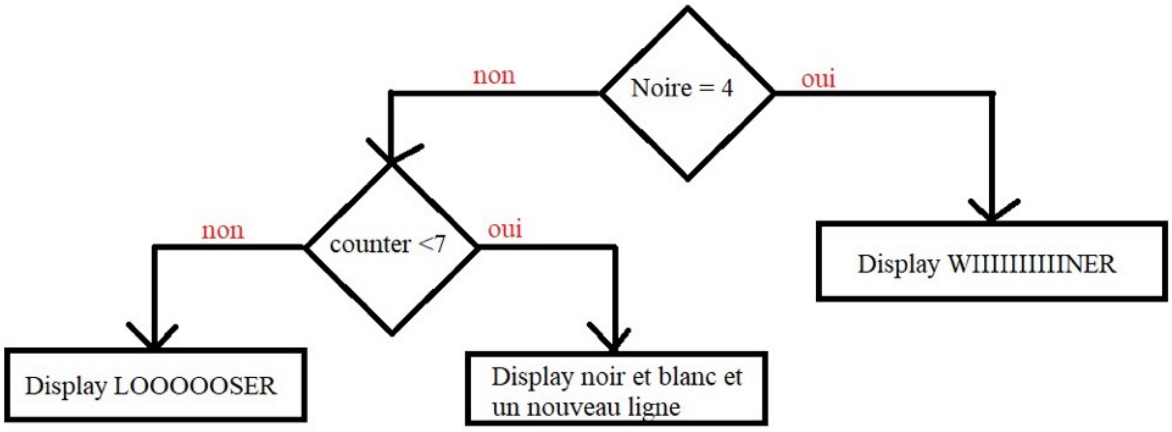



Le calcul de noir et blanc est fait par l'algorithme représente par cet organigramme

```
calculer_noir_et_blanc ;;;;;;;;IMportaNT;;;;;;;;algorithm pour
                                              ;; calculer Noir et blanc

      movf indf1,w    ;;cet algorithm consiste a comparer les 2 codes
      cpfseq indf0    ;;si il y a 2 chiffres qui sont les memes
                           ;;on compare les indices
      bra $+4         ;;meme indice --> noir = noir + 1
      call faire_incrementation  ;;different indice --> blanc = blanc + 1
      incf fsr0l,f
      movlw 0x04
      cpfseq fsr0l
      bra calculer_noir_et_blanc
      lfsr fsr0,0x300
      incf fsr1l,f
      movlw 0x14
      cpfseq fsr1l
      bra calculer_noir_et_blanc

      lfsr fsr1,0x310
      lfsr fsr0,0x300

      return

faire_incrementation
      movf fsr0l,w
      addlw 0x10
      cpfseq fsr1l
      bra $+6
      incf noir,f
      return
      incf blanc,f
      return
```

*Code montrant le calcul de noir et blanc*



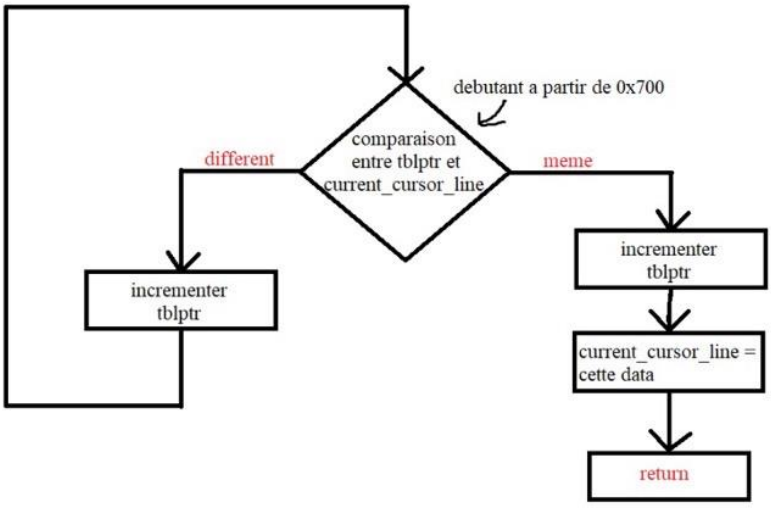

Cet organigramme qui dépend du nombre de noir et blanc montre les actions après que le joueur 2 appui enter et les 3 possibilités sont affichées ci-dessus

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 x 2 LCD | 80 | 81 | 82 | 83 | 84 | 85 | 86 through 8F |
| | C0 | C1 | C2 | C3 | C4 | C5 | C6 through CF |
| 20 x 1 LCD | 80 | 81 | 82 | 83 | through 93 | | |
| 20 x 2 LCD | 80 | 81 | 82 | 83 | through 93 | | |
| | C0 | C1 | C2 | C3 | through D3 | | |
| 20 x 4 LCD | 80 | 81 | 82 | 83 | through 93 | | |
| | C0 | C1 | C2 | C3 | through D3 | | |
| | 94 | 95 | 96 | 97 | through A7 | | |
| | D4 | D5 | D6 | D7 | through E7 | | |
| 40 x 2 LCD | 80 | 81 | 82 | 83 | through A7 | | |
| | C0 | C1 | C2 | C3 | through E7 | | |

Note: All data is in hex.

```
org 0x700
db 0xC0,0x94,0xD4,0x89,0xC9,0x9D,0xDD
```

A l'@ 0x700 (jusqu'à 0x706), la première colonne de chaque ligne va être sauvegarder dans une variable appelée current_cursor_line chaque fois on saute une ligne




*Algorithme pour le retour à la ligne* [1]

## 3.4-Affichage et délai

2 fonctions ont été utilisé pour l'affichage

```
COMNWRT  ;;la fonction qui controle le cursor du lcd

DATAWRT  ;;la fonction qui envoie la data vers le lcd
```

Pour les utiliser on a besoin de 3 autres fonctions de délai

1- La fonction Short delay qui doit être 450 ns
2- La fonction delay qui doit être entre 10 et 15 ms
3- La fonction long delay qui doit être entre 250 ms (pour initialiser l'écran LCD)

## 4-Réalisation

Pour la réalisation, il faut utiliser un oscillateur 4MHZ, donc on a utilisé la commande CONFIG OSC= XT avec le circuit suivant :

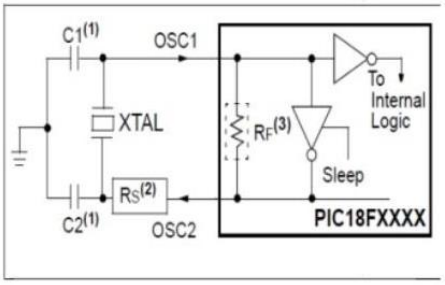
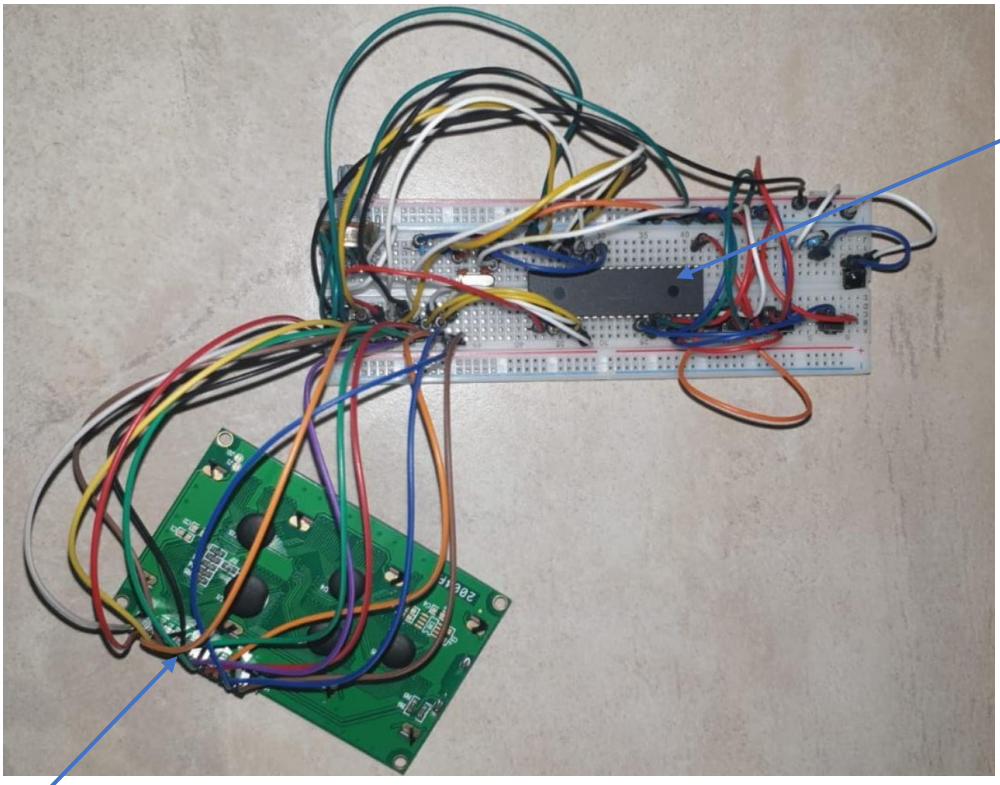



Figure 1

| Osc Type | Crystal Freq | Typical Capacitor Values Tested: | |
|---|---|---|---|
| | | C1 | C2 |
| LP | 32 kHz | 30 pF | 30 pF |
| XT | 1 MHz | 15 pF | 15 pF |
| | 4 MHz | 15 pF | 15 pF |
| HS | 4 MHz | 15 pF | 15 pF |
| | 10 MHz | 15 pF | 15 pF |
| | 20 MHz | 15 pF | 15 pF |
| | 25 MHz | 0 pF | 5 pF |
| | 25 MHz | 15 pF | 15 pF |

Tableau 1



## 5-Problèmes rencontrés

Tout en ayant pu créer le jeu Mastermind avec succès, nous avons rencontrés quelques difficultés comme choisir le même chiffre plusieurs fois dans la même combinaison ou avoir une combinaison aléatoire au début du jeu (sans intervention du joueur 1)

Enfin la réalisation du projet a l'aide des composants n'a pas abouti à un résultat final -> on n'a pas vu de résultats sur l'écran LCD.

## 6-Conclusion

Dans cet article, nous décrivons une réalisation du jeu Mastermind. Même si le jeu est bien connu et construit plusieurs fois, notre réalisation donne une approche passionnante à une programmation réalisée entièrement en langage de programmation assembleur.

Cette réalisation est particulièrement intéressante à des fins pédagogiques car elle offre une expérience passionnante

Une façon d'initier les étudiants à la programmation en assembleur et à l'architecture des processeurs pic18f.

## References

[1] M. A. Mazidi, PIC Microcontroller and Embeded Systems, New York : PEARSON Education , 2008.

## Remerciements

Nous exprimons nos vifs remerciements à notre professeur Dr. Chkaibane

Nous avons l'honneur de profiter et bénéficier de votre professionnalisme,

expériences, connaissances et remarques. Nous espérons que ce travail sera à la hauteur de vos attentes.

Et nous remercions aussi Dr. Sawma et Mr. Moughabghab pour leur aide à accomplir la réalisation.

Merci.