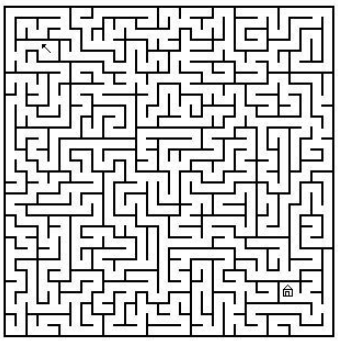
# Rapport de projet

# **Assembleur Projet N°5**

---

## Résolution d’un labyrinthe




Auteurs:

- LELAY Jules
- SECHEHAYE Benoit

ESIEE PARIS

# SOMMAIRE

# Introduction

Le but de cette unité nommée “Architecture” est de nous permettre d’avoir une approche globale de la conception des systèmes à base de microprocesseurs. L'objectif final est un projet en binôme dont le but est de faire fonctionner un EvalBot (figure ci-dessous). L’EvalBot devra faire des actions que le binôme aura décidé d’implémenter avec notamment l’utilisation des moteurs lui permettant d’avancer.

Image d’un EvalBot

L’EvalBot fonctionne grâce à un processeur LM3S9B92 de chez Texas Instrument.

# Présentation du projet

Le projet que notre binôme a décidé de faire sur l'EvalBot est la résolution d'un labyrinthe.

## Conventions

Il a été décidé en début de projet d'adopter un certain nombre de convention concernant la résolution de ce problème pour :

- Le simplifier
- Permettre une répartition juste des registres

### Les Directions

Tout d'abord, notons que l'orientation de l'EvalBot est faite suivant les axes cardinaux Ainsi, l'EvalBot peut être orienté vers le NORD, le SUD, l'EST et l'OUEST. Pour chacune de ces directions, il à été décider de leurs attribuer un doublé de valeur suivant le tableau suivant:

| Direction | Doublé de valeur |
|---|---|
| NORD | 0x00 |
| EST | 0x01 |
| SUD | 0x10 |
| OUEST | 0x11 |

## Le labyrinthe

Ensuite, à l'initialisation du programme, le robot aura déjà connaissance de la structure du labyrinthe. Il n'aura donc en aucun cas besoin de l'explorer pour en deviner le tracé.

Le labyrinthe est un quadrillage. (Exemple ci-dessous qui sera réutilisé par la suite)

Légende:

| Couleur | Signification |
|---|---|
| | Mur |
| | Robot |
| | Arrivé |

Nous partons également du postulat de base que le robot sera toujours orienté à l'initialisation vers le NORD. Le but de cette règle est d'éviter au robot de deviner dans quelle direction celui-ci est orienté par rapport au plan du labyrinthe dans le but de simplifier le démarrage du programme.

# Résolution du problème du labyrinthe

## Synthèse de l'algorithme

Pour résoudre le problème du labyrinthe, c’est-à-dire permettre au robot d’atteindre la case arriver, nous allons devoir dans un premier temps, à l’aide de la grille chargée en mémoire, déterminer l’un des chemins les plus court pour y arriver puis déplacer l’EvalBot jusqu'à la case d'arrivée en le faisant pivoter puis avancer dans la bonne direction.

Le programme se résume de la manière suivante :

**Debut**

**Initialisation du robot**
**Modification du plan permettre de trouver le chemin le plus court (1)**
**Tant que le robot n’est pas à la case arrivée (2)**
**Le robot cherche la case adjacente le rapprochant le plus de l’arrivée**
**Le robot pivote dans la direction de la case sur lequel avancer**
**Le robot avance vers la case qui le rapproche de l’arrivée**
**Fin tant que**

**Le robot avance une dernière fois pour sortir du labyrinthe**

**Fin**

## Chercher le plus court chemin (1)

Pour trouver le chemin le plus cours, on pars du schéma suivant où chaque type de case se voit attribuer une valeur par défaut :

- 0xFF pour un mur
- 0xFE pour une case libre
- 0x00 pour la case d'arrivée

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| FF | FE | FF | FE | FE | FE | FE | FE | FE | FF |
| FF | FE | FF | FE | FE | FE | FF | FF | FE | FF |
| FF | FE | FE | FE | FE | FE | FF | FF | FE | FF |
| FF | FF | FF | FF | FF | FE | FF | FF | FE | FF |
| FF | FF | FF | FF | FF | FE | FF | FE | FE | FF |
| FF | FF | FF | FF | FF | FF | FF | FE | FE | FF |
| FF | FE | FF | FE | FE | FE | FE | FE | FE | FF |
| FF | FE | FE | FE | FE | FE | FE | FE | FF | FF |
| FF | FF | FF | FF | 00 | FF | FF | FF | FF | FF |

En mémoire :

```
0x20000000: FF FF FF FF FF FF FF FF FF FF
0x2000000A: FF FE FF FE FE FE FE FE FE FF
0x20000014: FF FE FF FE FE FE FF FF FE FF
0x2000001E: FF FE FE FE FE FE FF FF FE FF
0x20000028: FF FF FF FF FF FE FF FF FE FF
0x20000032: FF FF FF FF FF FE FF FE FE FF
0x2000003C: FF FF FF FF FF FF FF FE FE FF
0x20000046: FF FE FF FE FE FE FE FE FE FE
0x20000050: FF FE FE FE FE FE FE FE FF FF
0x2000005A: FF FF FF FF 00 FF FF FF FF FF
```

On modifie la grille suivant le principe suivant:

**Toute case libre prend la valeur minimale entre leur valeur et celle de la case voisine de moindre valeur à laquelle on a rajouté 1.**
Jusqu'à ce que la grille soit à jour et ne connaisse plus de modification, on réitère le principe.
Cela nous donne la grille suivante :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| FF | 23 | FF | 17 | 16 | 15 | 14 | 13 | 12 | FF |
| FF | 22 | FF | 18 | 17 | 16 | FF | FF | 11 | FF |
| FF | 21 | 20 | 19 | 18 | 17 | FF | FF | 10 | FF |
| FF | FF | FF | FF | FF | 18 | FF | FF | 9 | FF |
| FF | FF | FF | FF | FF | 19 | FF | 7 | 8 | FF |
| FF | FF | FF | FF | FF | FF | FF | 6 | 7 | FF |
| FF | 5 | FF | 3 | 2 | 3 | 4 | 5 | 6 | FF |
| FF | 4 | 3 | 2 | 1 | 2 | 3 | 4 | FF | FF |
| FF | FF | FF | FF | 0 | FF | FF | FF | FF | FF |

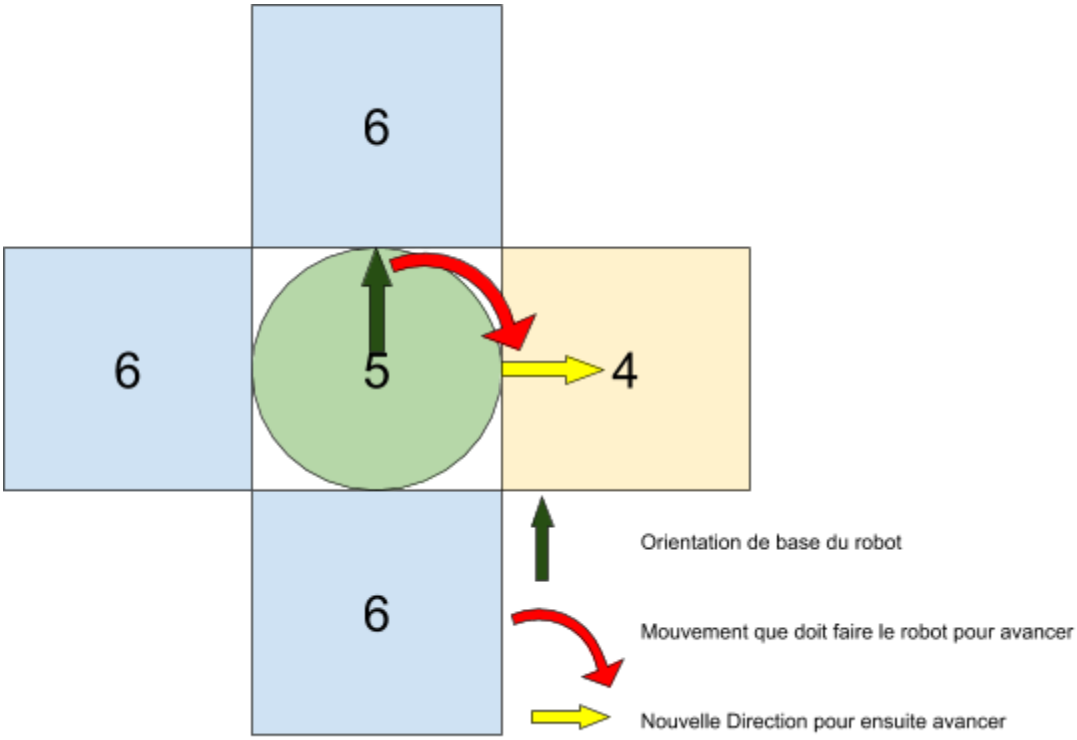
A partir de la case où est situé le robot il est possible de le déplacer pour atteindre l'arrivée rapidement.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| FF | 23 | FF | 17 | 16 | 15 | 14 | 13 | 12 | FF |
| FF | 22 | FF | 18 | 17 | 16 | FF | FF | 11 | FF |
| FF | 21 | 20 | 19 | 18 | 17 | FF | FF | 10 | FF |
| FF | FF | FF | FF | FF | 18 | FF | FF | 9 | FF |
| FF | FF | FF | FF | FF | 19 | FF | 7 | 8 | FF |
| FF | FF | FF | FF | FF | FF | FF | 6 | 7 | FF |
| FF | 5 | FF | 3 | 2 | 3 | 4 | 5 | 6 | FF |
| FF | 4 | 3 | 2 | 1 | 2 | 3 | 4 | FF | FF |
| FF | FF | FF | FF | 0 | FF | FF | FF | FF | FF |

Exemple de chemin possible pour atteindre l’arrivée

## Deplacer le robot (2)

Pour se déplacer, le robot va comparer la valeur de la case sur laquelle il se trouve avec les 4 cases adjacentes à lui pour trouver la plus petite. Il va ensuite comparer sa direction actuelle à celle qu’il devrait avoir pour pouvoir avancer vers la future case.
Le robot va pouvoir pivoter vers la future case pour être dans la bonne direction.
Enfin le robot va avancer vers la nouvelle case.




Exemple qui montre les actions du robot pour pouvoir se rapprocher de l'arrivée

# Implémentation de la solution dans l'EvalBot

## Attribution des registres

Pour au mieux gérer le robot et les informations qui lui sont nécessaire pour avancer, les différents registres de l'EvalBot (au nombre de 13) ont été attribué suivant le tableau suivant:

| Registre | Contenu |
|---|---|
| R0 | Valeur de la case sur lequel est le robot |
| R1 | Position X du robot |
| R2 | Position Y du robot |
| R3 | Orientation du robot |
| R4 | Valeur de la case que le robot doit atteindre |
| R5 | Position X de la case à atteindre par le robot par rapport à la case actuelle (CC2) |
| R6 | Position Y de la case à atteindre par le robot par rapport à la case actuelle (CC2) |
| R7 | Direction vers lequel le robot doit être orienté pour atteindre la nouvelle case |
| R8 a R11 | Buffers |
| R12 | Adressage |
| R13 | Stack Pointer |
| R14 | Link Register |
| R15 | Program Counter |

## Explication des choix de configuration des GPIO

Pour les GPIO, nous avons décidé d'utiliser les port D et H (contenu des ressources) pour le contenant les actions du moteur moteur et le port D pour le fichier contenant les fonctions utiles au programme.

- Port D activé : PIN 3 de CLK à l'état haut
- USER_SW1 utilisé :
    - PIN 6 du GPIODEN du Port D pour activer USER_SW1
    - PIN 6 du GPIOPUR du Port D à l'état haut pour mettre activer résistance de Pull-Up de USER_SW1

## Implémentation du labyrinthe en python

Pour initialiser et placer le robot dans la future grille, un script python a été créé dans le but de simplifier la création du tableau. Une chaîne de caractère est passée en paramètre, où chaque caractère représente une case de la grille.

| **Contenu de la case** | **Valeur initiale** | **Nouvelle valeur** |
|---|---|---|
| arrivée | 0 | 0x00 |
| vide | 1 | 0xFE |
| emplacement du robot | 2 | 0xFE |
| mur | 3 | 0xFF |

Les coordonnées seront stockées dans des variables à part et seront placées dans les registres correspondant en tant voulu.

Le scripte est le suivant:

```
# Représentation de la grille. 0 : arrivée ; 1 : libre; 2 : départ, 3 : mur
grille = """3333333333
3131111113
3231113313
3111113313
3333313313
3333313113
3333333113
3131111113
3111111133
3333033333"""

# Dictionnaire des valeurs de substitution
d = {'0' : '0x00', '1' : '0xFE' , '2' : "0xFE", '3' : "0xFF"}







def values(grille):
    lines = grille.split("\n")
    x=0
    y=0
    largeur = len(lines[0])
    hauteur = len(lines)
    l = []
    for line in lines:
        for c in line:
            l.append(d[c])
            if c == "2":
                x = line.index(c)
                y = lines.index(line)
    cases = ", ".join(l)

    s = """        AREA    CONSTANTES, DATA, READONLY
X       DCB        {0}
Y       DCB        {1}
LARGEUR DCB        {2}
HAUTEUR DCB        {3}
        AREA    GRILLE, DATA, READWRITE
CASES   DCB        {4}""".format(x,y,largeur,hauteur,cases)
```

```
    print(s)

if __name__ == "__main__":
    values(grille)
```

## Les fichiers utilisés

### Moteur.s

Ressource fourni par l'établissement.
Auteurs : M. Akil, T. Grandpierre, R. Kachouri : département IT - ESIEE Paris - 01/2013 - Evalbot (Cortex M3 de Texas Instrument)

### Main.s

Fichier implémentant la boucle principale du programme.

### Fonctions.s

Fichier source implémentant les fonctions utilisées par le programme :

- SW_INIT
  Permet d'initialiser et d'activer les switchs de l'EvalBot
- INIT
  Permet d'initialiser l'ensemble des fonctionnalités utiles à l'EvalBot pour le programme (regroupant la fonction précédente et MOTEUR_INIT)
- MOTEUR_PIVOTE_DROITE
  Permet de faire pivoter l'EvalBot de 90° dans le sens horaire.
- MOTEUR_PIVOTE_GAUCHE
  Permet de faire pivoter l'EvalBot de 90° dans le sens anti-horaire.
- MOTEUR_DEMI_TOUR
  Permet de faire pivoter l'EvalBot de 180° dans le sens horaire.
- MOTEUR_AVANCE

Permet de faire avancer l’EvalBot d’une case en ligne droite

- CHOIX_DIRECTION

  Permet de déterminer dans quelle direction doit se diriger l’EvalBot pour se rapprocher de l’arrivée.

- SOLVE_MAZE

  A partir du plan du labyrinthe en mémoire, la fonction le modifie pour déterminer le chemin le plus court (voir “Chercher le plus court chemin (1)”)

## Values.py

Simple formatage des données de cartographie du labyrinthe développé plus tôt