PROJET 2

SIMULATEUR DE MICROPROCESSEUR 6809

Réalisés par
Douae AMZIL
Chaimaa ALIOUAN
Hassan AMRANI
Hasnae EL JABARI

Encadré par monsieur
Hicham BENALLA

PLAN:

1. Introduction
2. la fenêtre principale
3. Classe Registre et Register Display
4. Classe Flag
5. Classe Rom et Rom Display
6. Classe RAM Display
7. Classe Instruction
8. Classe UAL
9. Exemple de simulation

introduction

MOTO6809 c’est un logiciel qui permet de simuler le fonctionnement du microprocesseur 6809 de Motorola facilitant ainsi débogage des programmes écrits en langage d’assemblage 6809 .

Ses caractéristiques:

- **Visualisation simultanée du contenu de tous les registres internes**
- **Fenêtres flottantes pour la RAM et la ROM**
- **Fenêtre flottante présentant 20 lignes de programme**
- **Point d'arrêt**
- **Modes Pas-à-pas, Exécution complète**
- **Editeur intégré.**

fenêtre pricipale

la fenêtre principale est titulé par moto 6809 S , elle comporte deux panels :

une panel à droite qui comporte le texte éditeur où on va écrire notre programme

une panel à gauche qui contient l’architecture interne du 6809 et le contenu de ses différents registres internes lors de la simulation associé avec une autre panel qui comporte une barre des boutons

barre des boutons

- **new file :** Permet de créer un nouveau fichier assembleur
- **load file :** charger un fichier
- **Save:** Enregistrer le programme assembleur actuellement ouvert
- **Execute :** Exécuter le programme assembleur
- **Pas à pas :** Fait avancer le programme assembleur d'une instruction
- **Programme :** Permet d'afficher ou de cacher la fenêtre programme
- **RAM :** Permet d'afficher ou de cacher la fenêtre RAM
- **ROM :** Permet d'afficher ou de cacher la fenêtre ROM
- **RESET:** Permet de réinitialiser le microprocesseur

Nous avions crée une classe MainFrame qui représente la fenêtre principale .Elle comprend comme

ATTRIBUTS

ual (type UAL) : Instance de la classe UAL représentant l'Unité Arithmétique et Logique.
Inst (type Instruction) : Instance de la classe Instruction pour manipuler des instructions.
originalFile (type File) : Fichier original chargé dans l'application ou crée.
textArea (type JTextArea) : Zone de texte pour afficher le contenu du programme ou pour entrer des instructions.
memoryROMDisplay (type MemoryROMDisplay) : Fenêtre d'affichage de la mémoire ROM.
memoryRAMDisplay (type MemoryRAMDisplay) : Fenêtre d'affichage de la mémoire RAM.
panel (type JPanel) : Panneau contenant les boutons et éléments de contrôle de l'interface.
stopReading (type volatile boolean) : Indicateur de lecture utilisé pour l'exécution pas à pas.

METHODES PRINCIPALES

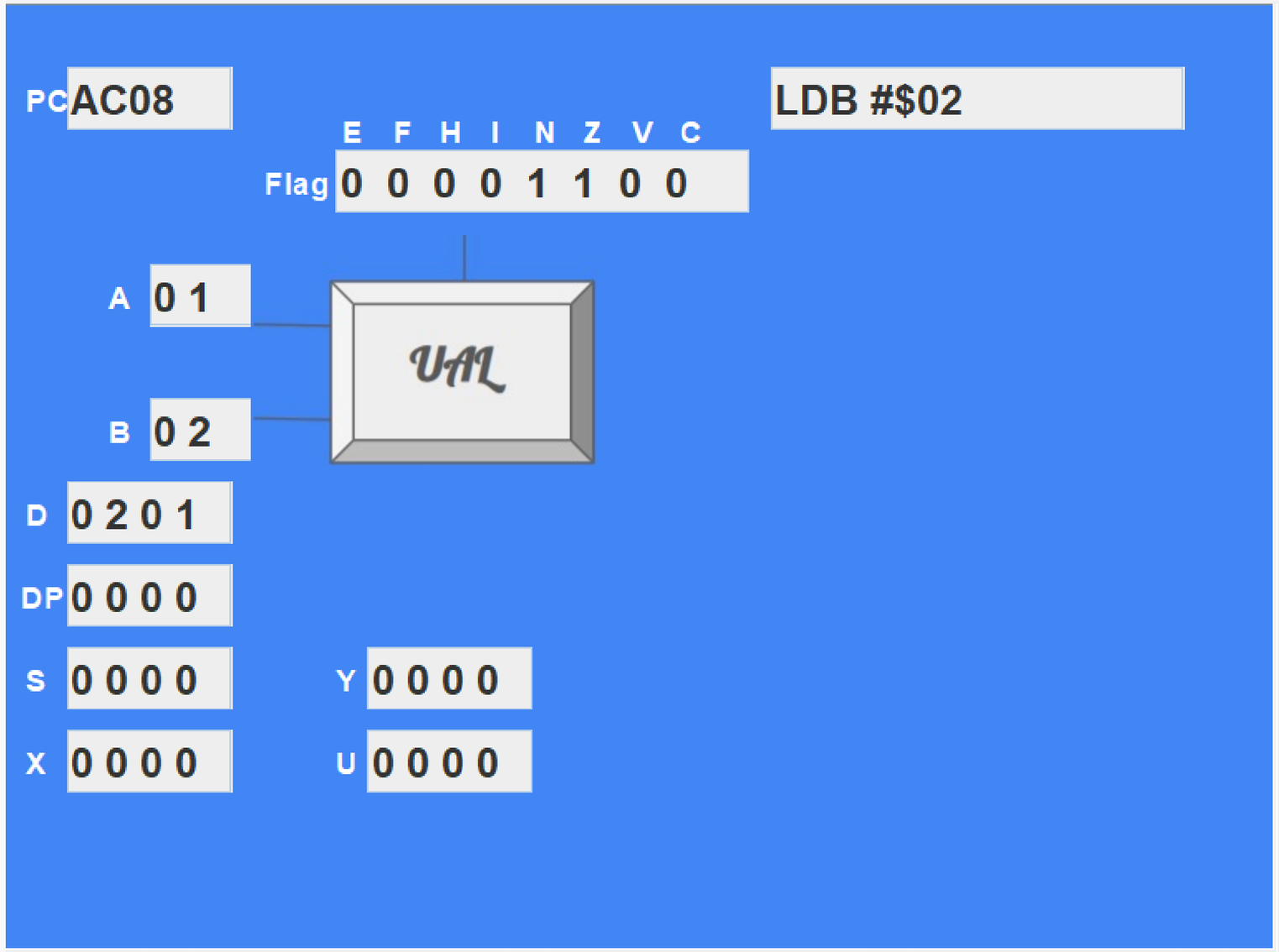
mainwindow() : Méthode pour créer et afficher la fenêtre principale de l'application.Elle contient une zone de texte, et crée des boutons pour les opérations telles que charger, enregistrer, exécuter, afficher le programme...
saveToFile(String content, File file) : Méthode pour Sauvegarder le contenu de la zone de texte dans un fichier spécifié.
showMessage(String message) :Affiche une fenêtre modale avec un message spécifié.
loadFile(JTextArea textArea) : Charge le contenu d'un fichier dans la zone de texte.
createAndSaveFile(JTextArea textArea) : Crée un nouveau fichier et sauvegarde son contenu.
Execute: Une classe interne pour l'exécution du contenu du fichier chargé.
PasPasButton : Une classe interne pour le bouton "Pas à Pas", permettant de traiter le fichier instruction par instruction.
ResetButton : Une classe interne pour le bouton de réinitialisation, qui réinitialise certains états et permet une nouvelle exécution.

LA CLASSE MAIN ▼

c’est la classe qui contient la méthode main de notre programme:

```
public class MainMoto6809s
{               public static void main(String[] args)
                        {
                                MainFrame.mainwindow();
                         }
}
```

MOTO 6809 S



;éditeur de texte
LDA #$01
LDB #$02

Load File | New File | save | Execute | Programme

ROM | RAM | Pas Ã Pas | Reset

Classe Registre

La classe Registe est une classe qui représente un ensemble de registres d'une unité de traitement.

La classe est utilisée pour stocker et partager des informations sur l'état du microprocesseur entre différentes parties du programme.

Elle fournit des méthodes pour manipuler les registres de l'unité de traitement en ce qui concerne l’initialisation , la modification et l’affichage.

Ses Attributs

- ***line: Une chaîne de caractères représentant une ligne du programme en cours d'exécution.***
- ***accumulatorA: Une chaîne de caractères représentant le contenu de l'accumulateur A.***
- ***accumulatorB: Une chaîne de caractères représentant le contenu de l'accumulateur B.***
- ***AccumulatorD: Une chaîne de caractères représentant le contenu de l'accumulateur D, qui est décomposé en accumulateurs A et B.***
- ***indexX: Une chaîne de caractères représentant le contenu du registre d'index X.***
- ***indexY: Une chaîne de caractères représentant le contenu du registre d'index Y.***
- ***RPS: Une chaîne de caractères représentant le registre d'état du processeur (statut).***
- ***RPU: Une chaîne de caractères représentant le registre d'état du processeur (user).***
- ***ProgramCounter: Une chaîne de caractères représentant le compteur de programme, indiquant la ligne du programme actuellement en cours d'exécution.***

Ses Méthodes

Des paires de méthodes sont fournies pour chaque registre afin de récupérer et définir leurs valeurs, suivant la convention des getters et setters.
Les méthodes get permettent de récupérer la valeur actuelle des registres.
Les méthodes set permettent de définir de nouvelles valeurs pour les registres.

Registrer DISPLAY

La classe RegisterDisplay est une classe graphique héritant de JPanel qui représente l'affichage des registres d'une unité de traitement.
son rôle consiste sur les apparences graphiques de différents registres .

Elle utilise la classe Registre pour accéder aux valeurs des registres.

Ses Attributs

*Utilisation **des champs de texte (JTextField)** pour afficher les valeurs des différents registres de l'unité de traitement, **tels que le compteur de programme (PC), les accumulateurs (A, B, D), les indices (X, Y), les indicateurs de statut, etc.***
***Une instance de la classe Flag** pour gérer et afficher les drapeaux.*

Ses Méthodes

- **registerpanel()**: Méthode pour configurer la disposition, les dimensions et le style visuel de l'affichage des registres.

- **addLabel(String labelText, int x, int y)**: Méthode utilisée par **registerpanel()** pour ajouter une étiquette avec un texte spécifié et la positionner à des coordonnées spécifiques.

- **Des méthodes publiques statiques** (**setPC**, **setline**, **setflag**, **setA**, **setB**, **setD**, **setS**, **setX**, **setU**, **setY**) sont fournies pour mettre à jour les valeurs des registres depuis d'autres parties du programme.

FLAG

la classe **Flag** représente un ensemble de drapeaux (flags) utilisés pour stocker des informations sur l'état interne d'une unité de traitement.

cette classe encapsule la gestion des drapeaux et fournit des méthodes pour définir, obtenir et afficher l'état de ces drapeaux

les drapeaux utilisés sont :

- **carryFlag**: Drapeau de retenue.
- **zeroFlag**: Drapeau zéro, indiquant si le résultat d'une opération est zéro.
- **irqDisableFlag**: Drapeau de désactivation des interruptions.
- **halfCarryFlag**: Drapeau de demi-retenue.
- **interruptFlag**: Drapeau d'interruption.
- **negativeFlag**: Drapeau de négativité, indiquant si le résultat d'une opération est négatif.
- **overflowFlag**: Drapeau de dépassement.
- **extendedFlag**: Drapeau étendu.

MÉTHODES

SetAllFlagsFalse() : Initialise tous les drapeaux à la valeur false.

Méthodes de définition des drapeaux (setters):

- Des méthodes sont fournies pour définir individuellement chacun des drapeaux en spécifiant une valeur booléenne (setCarryFlag, setZeroFlag, ..., setExtendedFlag).

Méthodes d'obtention des drapeaux(getters) :

- Des méthodes sont fournies pour obtenir la valeur actuelle de chaque drapeau (getCarryFlag, getZeroFlag, ..., getExtendedFlag).

Affichage des drapeaux :

- La méthode **displayFlags()** renvoie une chaîne de caractères représentant l'état actuel de tous les drapeaux. Cette chaîne est formatée de manière à indiquer la valeur de chaque drapeau avec un "1" pour activé et un "0" pour désactivé

ROM

la classe ROM permet de représenter et gérer la mémoire ROM (Read-Only Memory) de notre logiciel.

la classe Rom a pour objectif principal d'initialiser et de fournir un accès aux données de la mémoire ROM , en utilisant une structure de données HashMap pour stocker les informations associées à chaque adresse mémoire.

Attributs :

- **cpt** (statique) : Un compteur statique
- **word** (statique, ArrayList) : Une liste statique qui stocke des instructions ou des données pour la mémoire ROM.
- **startAddress** (statique) : Adresse de début de la mémoire ROM.
- **endAddress** (statique) : Adresse de fin de la mémoire ROM.
- **PC** (statique, ArrayList) : Une liste statique qui stocke des valeurs liées au compteur de programme (Program Counter).

Méthodes :

- **RomInitial()** : Initialise et retourne un objet **HashMap** représentant les données de la mémoire ROM avec des adresses comprises entre **startstaticAddress** et **endstaticAddress**. Les données sont initialement définies à "FF".
  - Les adresses sont générées et les valeurs "FF" sont associées à chaque adresse.
- **getROMdata()** (statique) : Obtient les données de la mémoire ROM, en utilisant les données initialisées par **RomInitial()** et en les remplaçant par les valeurs stockées dans la liste statique **word**.
  - Les adresses de la mémoire sont mappées aux valeurs correspondantes stockées dans **word**.
  - La méthode retourne un objet **HashMap** représentant les données mises à jour de la mémoire ROM.

ROM DISPLAY

la classe MemoryRomDisplay est une classe qui conçue pour afficher le contenu de la mémoire ROM

la classe MemoryROMDisplay fournit une fenêtre d'affichage pour la mémoire ROM, organisant et affichant les données de manière claire et organisée dans une interface utilisateur graphique.

Attributs :

- **displayArea** (de type **JTextArea**) : Composant graphique destiné à afficher le contenu de la mémoire ROM.
- **memoryMap** (de type **Map<String, String>**) : Une carte associant des adresses de la mémoire ROM à leurs valeurs correspondantes.

Constructeur :

- **MemoryROMDisplay()** : Initialise la classe en récupérant les données de la mémoire ROM depuis la classe **Rom** et en appelant la méthode **initializeUI()**.

Méthode initializeUI :

Initialise l'interface utilisateur.

- Titre de la fenêtre défini comme "ROM".
- Crée un panneau avec une mise en page BorderLayout.
- Utilise un composant **JTextArea** pour afficher le contenu de la mémoire ROM dans une zone de défilement.
- Ajoute un bouton "Go to Top" pour faire défiler vers le haut.
- Met à jour l'affichage en appelant la méthode **updateDisplay()**.
- Affiche la fenêtre avec une taille fixe, non redimensionnable, et sans décorations.

Méthode updateDisplay :

Met à jour le contenu affiché dans la zone de texte.

- Efface le contenu précédent.
- Trie la carte **memoryMap** par clé (adresse) en utilisant la méthode **sortHashMapByKey**.
- Affiche chaque entrée de la mémoire ROM dans la zone de texte.

SortHashMapByKey(Map<K, V> map)

méthode statique qui Trie une carte (**Map**) par clé (adresse) et renvoie une nouvelle carte triée.

- Utilise une liste temporaire pour stocker les entrées de la carte.
- Trie la liste en fonction des adresses.
- Recrée une nouvelle carte triée à partir de la liste.

RAM DISPLAY

La classe **MemoryRAMDisplay** consiste pour afficher graphiquement le contenu de la mémoire RAM de notre logiciel.

Elle fournit une interface graphique simple pour afficher et visualiser le contenu de la mémoire RAM .

Constructeur :

- La classe a un constructeur par défaut (**MemoryRAMDisplay()**) qui initialise l'interface utilisateur en appelant la méthode **initializeUI()**.

Variables statiques :

- La classe contient des variables statiques, notamment **fromual** qui est une HashMap statique.

Méthode RamInitial :

- Méthode statique qui génère et retourne une mémoire RAM initiale sous forme de **HashMap<String, String>**. La RAM est initialisée avec des adresses hexadécimales allant de "0000" à "03FF" et des valeurs par défaut "00".

Méthode initializeUI :

- Initialise l'interface utilisateur de la fenêtre.
- Crée une fenêtre Swing avec une zone de texte, une barre de défilement et un bouton "Go to Top" pour remonter en haut de la mémoire.
- Appelle la méthode **updateDisplay** pour afficher la mémoire RAM initiale.

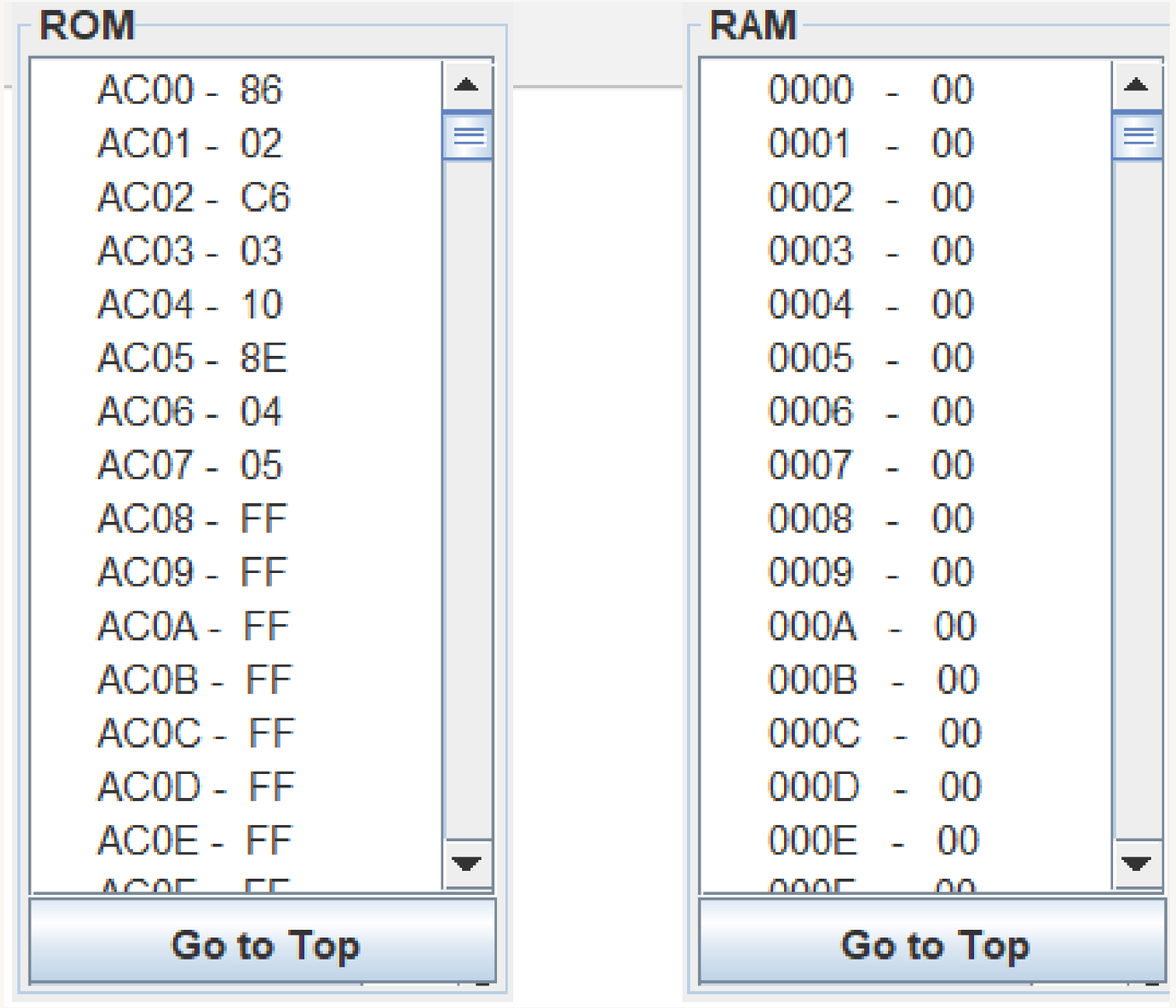
Méthode updateDisplay :

- Met à jour l'affichage de la mémoire RAM avec les données fournies en paramètre.
- Les données initiales de la RAM sont modifiées selon les données de mise à jour.
- Trie et affiche le contenu mis à jour dans la zone de texte.

Méthode sortHashMapByKey :

- Méthode générique pour trier une **HashMap** par clé et retourner une **LinkedHashMap** triée.

les fenêtres de la RAM et la ROM



Instruction

La classe **Instruction** permet de représenter des instructions processeur et effectuer des opérations associées sur les registres d'une machine

la classe **Instruction** encapsule la logique de manipulation des instructions processeur et des registres associés dans un système, fournissant des méthodes pour charger des valeurs, effectuer des opérations arithmétiques, et gérer certaines opérations spécifiques.

Attributs

- un objet de la classe **Registre**,
- un objet de la classe **Flag**
- Une table de hachage **codeOpcmd** qui associe les mnémoniques des instructions aux codes opérationnels .

Méthodes de transfert de données :

La classe propose des méthodes pour effectuer des transferts de données entre les registres, tels que **LDA** (Load Accumulator A), **LDB** (Load Accumulator B), **LDD** (Load Accumulator D), etc.

Opérations logiques :

La classe prend en charge des opérations logiques telles que **ANDA** (Logical AND with Accumulator A), **EORA** (Logical XOR with Accumulator A), **EORB ,**etc.

Méthodes de manipulation de bits :

Méthodes telles que **COMA, COMB, INCA, DECA**, etc., pour effectuer des opérations de manipulation de bits.

Méthodes pour les opérations arithmétiques :

ADDA(String menu), **ADDB**(String menu): Additionne une valeur hexadécimale à l'accumulateur A ou B.

Méthodes de manipulation des données :

Elle fournit des méthodes pour effectuer des opérations sur les données, telles que **la conversion entre hexadécimal et binaire, ainsi que des opérations de complément à un.**

Méthodes de stockage :

La classe offre des méthodes pour stocker les données des registres dans la mémoire RAM, telles que **STA** (Store Accumulator A) et **STB** (Store Accumulator B).

Gestion des exceptions :

La classe gère les erreurs en affichant des messages d'erreur via JOptionPane et en appelant la méthode SWI pour arrêter la lecture des instructions.

Interaction avec l'interface utilisateur :

La classe met à jour l'interface utilisateur, notamment en modifiant l'affichage des registres en Utilisant une classe **RegisterDisplay**

UAL

La classe **UAL** (Unité Arithmétique et Logique) est responsable du traitement des instructions et de la communication avec les composants mémoire (ROM et RAM).

Elle effectue des opérations de base sur les registres en fonction des instructions du processeur.
Elle utilise les classes Registre et Instruction pour accéder et manipuler les registres, ainsi que pour obtenir les codes opérationnels des instructions.

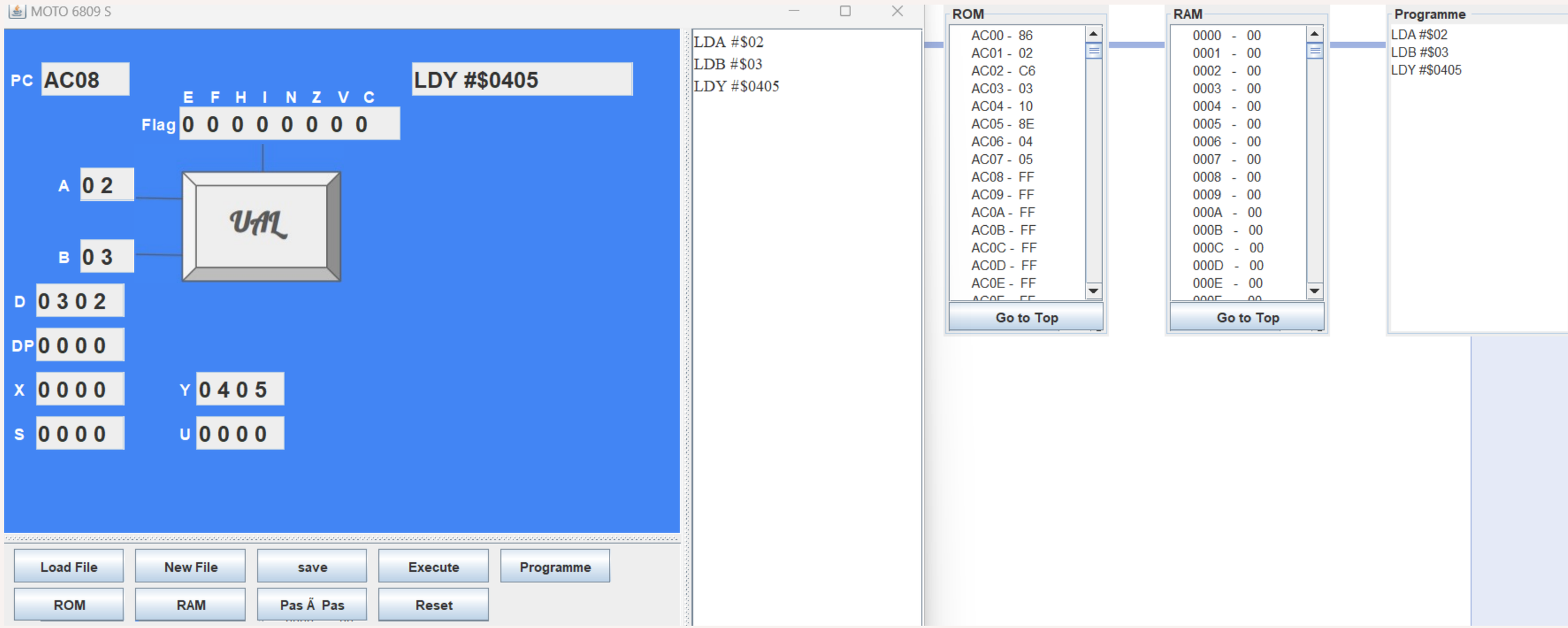
Attributs

- **R** (de type **Registre**) : Instance de la classe **Registre** pour gérer les registres du processeur.
- **Inst** (de type **Instruction**) : Instance de la classe **Instruction** pour effectuer les opérations d'instructions.
- **Torom (de type ArrayList<String>) :** Liste pour stocker les données du ROM.

Méthodes

- **verifierhex(String input)** : Vérifie si une chaîne de caractères est au format hexadécimal valide.
- **reforme(String input)** : Reformate une chaîne hexadécimale pour l'affichage.
- **Sendromddata(ArrayList torom)** : Envoie les données du ROM (Read-Only Memory) à une interface utilisateur,utilisée pour afficher le contenu du ROM.
- **Traitement(String line)** : Traite une ligne d'instruction, effectue des opérations sur le ROM, et met à jour l'affichage des registres.
  - Analyse la ligne d'instruction.
  - Gère différents modes d'adressage et de formats d'instructions.
  - Effectue des opérations sur les registres en fonction de l'instruction et des opérandes.
  - Gère des cas d'erreur et affiche des messages d'erreur à l'aide de JOptionPane.

EXEMPLE DE SIMULATION



Thanks