

# GUIDE D'UTILISATION SIMULATEUR MOTOROLA 6809

Créé par :

AMELLACHOU SALMA

OMAYMA AIT KHAM

Encadrée par :

Monsieur. Hicham BENALLA

Le 21 décembre 2025

# Sommaire :

1 Introduction .....	2
2 Présentation du MC6809 .....	3
2 .1 Registres Internes.....	3
3 Présentation du simulateur .....	5
3.1 Interface Graphique Globale.....	5
4 Instructions implémentées et modes d'adressage .....	6
4.1 Instructions .....	6
4.2 Modes d'adressage .....	6
5 Fonctionnement du simulateur .....	7
5.1 Édition et Compilation .....	7
5.2 Contrôle de l'exécution .....	8
6 Débogueur et fonctionnalités avancées .....	8
6.1 Visualisation de la RAM.....	8
6.2 Visualisation de la ROM .....	9
6.3 Modification des Registres .....	10
7 Tests et validation.....	11
7.1 Tests Unitaires .....	11
7.2 Tests d'Intégration.....	12
8 Conclusion .....	13
8.1 Limites .....	13
8.2 Améliorations possibles .....	13

# Introduction

Ce projet porte sur la réalisation d'un simulateur du microprocesseur Motorola 6809. Il s'agit d'un programme permettant de reproduire le fonctionnement de ce microprocesseur à travers une émulation logicielle.

Il existe afin de faciliter la compréhension du fonctionnement interne du MC6809, sans avoir recours à un matériel réel. Le simulateur permet d'observer l'exécution des instructions, l'évolution des registres et l'utilisation de la mémoire, ce qui est particulièrement utile dans un cadre pédagogique.

L'application développée permet aux étudiants et aux développeurs d'écrire des programmes en langage assembleur, de les compiler, et d'observer en temps réel l'évolution de la mémoire et des registres internes du processeur.

# Présentation du MC6809

Le Motorola 6809 est un microprocesseur 8/16 bits développé par Motorola. Il est reconnu pour son architecture avancée et son jeu d'instructions puissant, ce qui en fait un processeur largement utilisé dans les systèmes embarqués et dans l'enseignement de l'architecture des microprocesseurs.

## Registres Internes

L'architecture du 6809 repose sur un ensemble de registres spécifiques que nous avons reproduits fidèlement dans notre simulateur.

Registres							
O	O	O	O	O	O	O	O
E	F	H	I	N	Z	V	C
PC							
0000							
A				B			
00				00			
D				X			
0000				0000			
Y				S			
0000				0000			
U				DP			
0000				00			

**Figure 1** : Vue détaillée des registres du simulateur

Comme illustré dans la **Figure 1** , le processeur contient :

- **PC (Program Counter)** : Compteur ordinal de 16 bits (affiché ici à 0000). Il pointe vers la prochaine instruction à exécuter.
- **A et B** : Deux accumulateurs de 8 bits, utilisés pour les opérations arithmétiques et logiques. Ils peuvent être concaténés pour former le registre **D** (16 bits).

- **X et Y** : Registres d'index de 16 bits, utilisés pour les modes d'adressage indexés.
- **S et U** : Pointeurs de pile (S pour le système, U pour l'utilisateur).
- **DP (Direct Page)** : Registre 8 bits définissant la page mémoire active pour l'adressage direct, c'est la combinaison des accumulateurs A et B.
- **CC (Condition Code)** : Représenté par les drapeaux en haut de l'image , Il contient plusieurs indicateurs appelés flags(**E, F, H, I, N, Z, V, C**), qui reflètent le résultat des opérations effectuées par le processeur .

- **flags** :

**E** : sauvegarde complète du processeur lors d'une interruption.

**F** : masque les interruptions rapides (FIRQ).

**H** : signale une retenue entre bits 3 et 4.

**I** : masque les interruptions standard (IRQ).

**N** : indique un résultat négatif.

**Z** : indique un résultat nul.

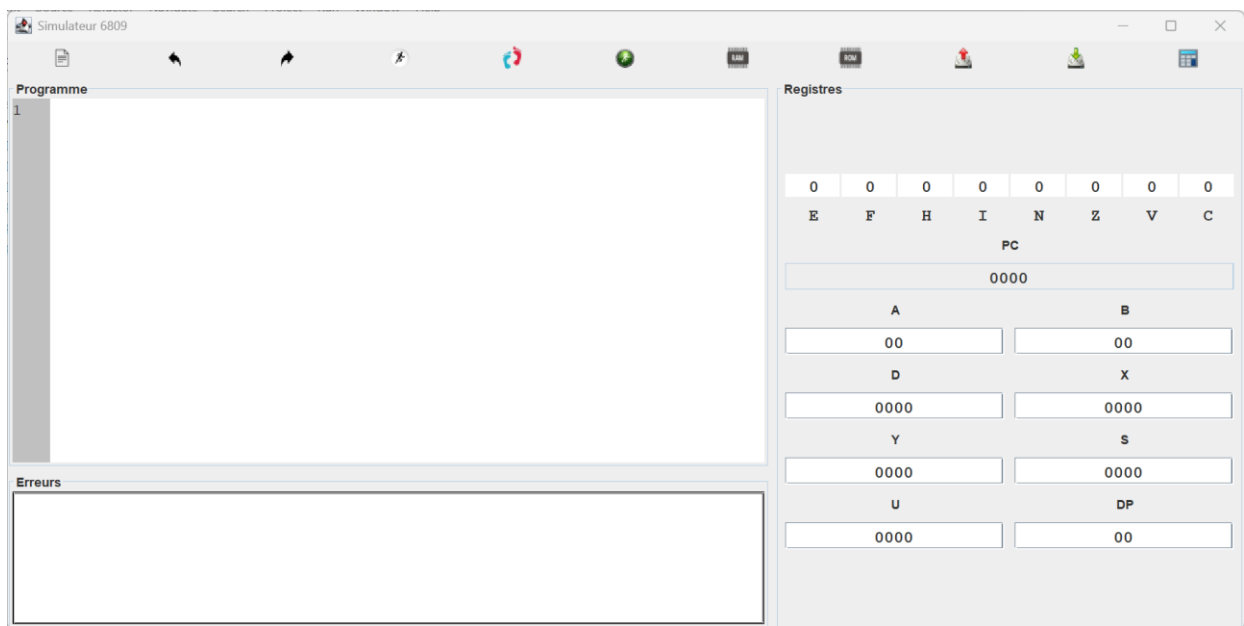
**V** : signale un dépassement arithmétique signé.

**C** : indique une retenue ou un emprunt.

# Présentation du simulateur

## Interface Graphique Globale

L'interface utilisateur a été conçue pour être intuitive et regrouper toutes les informations nécessaires sur un seul écran.



**Figure 2 :** Interface principale du simulateur

Comme le montre la **Figure 2** , l'application se divise en trois zones principales :

1. **La barre d'outils (Haut)** : Contient les commandes d'action .
2. **Zone de gauche (Programme et Erreurs)** : Dédiée à l'édition du code source et au retour du compilateur .
3. **Zone de droite (Registres)** : Affiche l'état interne du CPU .

Cette disposition permet à l'utilisateur de coder à gauche et de voir immédiatement l'effet sur le processeur à droite et les erreurs en bas .

# Instructions implémentées et modes d'adressage

Le simulateur prend en charge un sous-ensemble significatif du jeu d'instructions du 6809 pour permettre l'exécution d'algorithmes complexes.

## 1. Instructions

- **Transfert** : LDA, LDB, STA, STB, LDX, STX (Chargement et stockage).
- **Arithmétique** : ADDA, ADDB, SUBA, SUBB, MUL (Addition, Soustraction, Multiplication).
- **Pile** : PSHS, PULU, PULS, PULU (Empiler et dépiler).

## 2. Modes d'adressage

Le simulateur gère :

- **Immédiat (#\$VAL)** : La valeur est dans l'instruction.
- **Direct (\$AD)** : Adresse courte sur la page définie par DP.
- **Indexé (ex : ,X )** : Adresse calculée par rapport à un registre d'index.

# Fonctionnement du simulateur

L'utilisateur interagit principalement via la zone de programmation et la barre d'outils.

## 1. Édition et Compilation

La zone de gauche est le cœur de l'interaction textuelle.



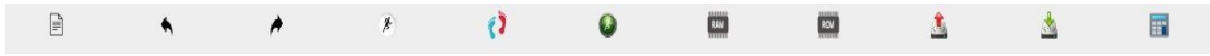
**Figure 3 :** Zone d'édition de programme et console d'erreurs

Sur la **Figure 3**, on distingue :

- **La zone Programme** : L'utilisateur saisit ici son code assembleur. Une colonne grise sur la gauche indique les numéros de ligne pour faciliter le repérage.
- **La zone Erreurs** : Située en bas, cette console affiche les messages du compilateur. Si une instruction est mal formée (ex : "LDZ" qui n'existe pas), le message apparaîtra ici en rouge, empêchant l'exécution.

## 2. Contrôle de l'exécution

La barre d'outils permet de piloter le flux du programme.



**Figure 2** : Barre d'outils de contrôle

La **Figure 2** présente les icônes disponibles :

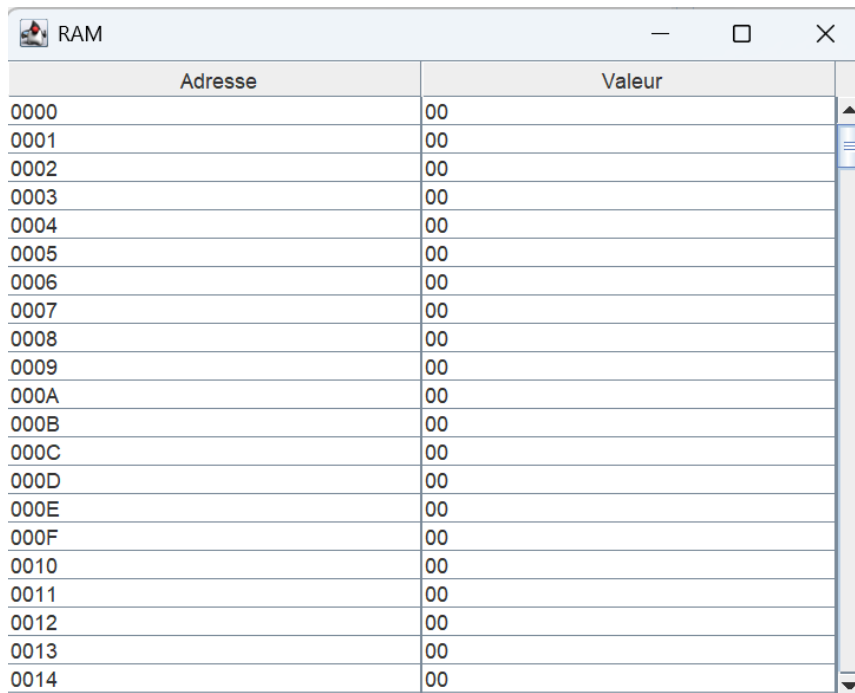
- **Nouveau (Feuille)** : Réinitialise les registres.
- **Undo/Redo (Flèches)** : Annule ou rétablit les modifications de texte.
- **Manipulation(homme qui court)** : Lance la compilation du code .
- **Pas à pas (Empreintes)** : Exécute une seule instruction à la fois.
- **Exécution (Puce verte)** : Lance le programme en continu.
- **Afficher RAM** : Lancer une fenêtre qui affiche la table RAM.
- **Afficher ROM** : Lancer une fenêtre qui affiche la table ROM.
- **Ouvrir un fichier** : Ouvrir un fichier dans le disque.
- **Sauvegarder le code** : Sauvegarder le code dans un fichier dans le disque.
- **Calculatrice** : Aide à vérifier le résultat avec des opérations en hexadécimal .

# Débogueur et fonctionnalités avancées

Pour analyser le comportement d'un programme, le simulateur offre des outils de visualisation de la mémoire.

## 1. Visualisation de la RAM

La mémoire vive (RAM) est simulée sous forme d'un tableau adressable.



The screenshot shows a window titled "RAM" with a table of memory addresses and values. The table has two columns: "Adresse" and "Valeur". The addresses range from 0000 to 0014, and the values are all 00. The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

Adresse	Valeur
0000	00
0001	00
0002	00
0003	00
0004	00
0005	00
0006	00
0007	00
0008	00
0009	00
000A	00
000B	00
000C	00
000D	00
000E	00
000F	00
0010	00
0011	00
0012	00
0013	00
0014	00

**Figure 3** : Fenêtre de visualisation de la RAM

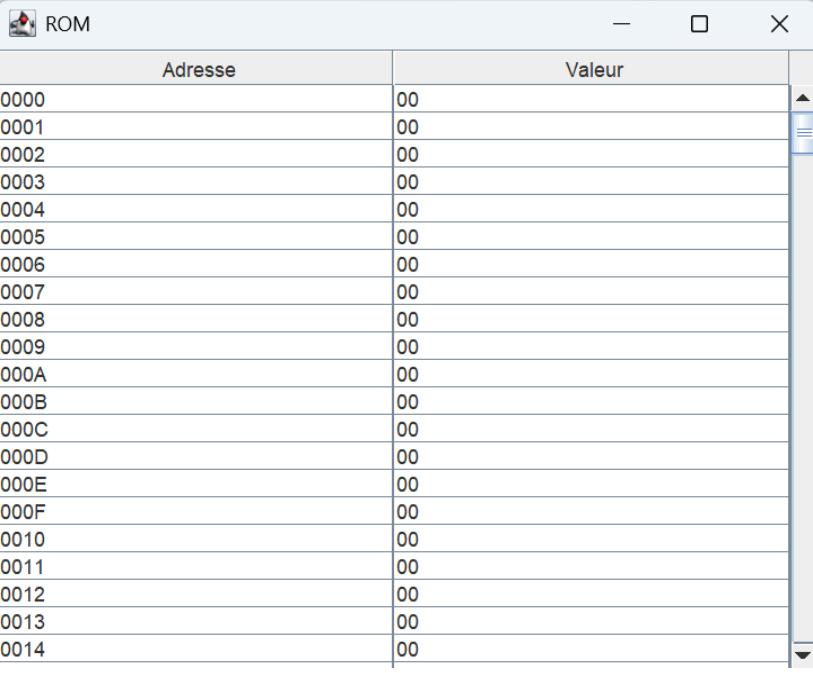
L'utilisateur peut ouvrir la fenêtre ci-dessus **Figure 3** via le bouton "RAM" de la barre d'outils. Elle affiche :

- **Colonne Adresse** : L'adresse mémoire hexadécimale (0000, 0001...).
- **Colonne Valeur** : Le contenu de l'octet à cette adresse.

Cette vue est mise à jour dynamiquement lorsqu'une instruction est exécutée. L'utilisateur peut modifier les valeurs dans chaque case de la RAM.

## 2. Visualisation de la ROM

De la même manière, la mémoire morte (ROM) peut être inspectée.



The screenshot shows a window titled 'ROM' with a table containing two columns: 'Adresse' and 'Valeur'. The table lists addresses from 0000 to 0014, with hexadecimal values '00' for each. A vertical scrollbar is visible on the right side of the table.

Adresse	Valeur
0000	00
0001	00
0002	00
0003	00
0004	00
0005	00
0006	00
0007	00
0008	00
0009	00
000A	00
000B	00
000C	00
000D	00
000E	00
000F	00
0010	00
0011	00
0012	00
0013	00
0014	00

**Figure 4 :** Fenêtre de visualisation de la ROM

La **Figure 4** montre le contenu de la ROM. Bien qu'elle soit en lecture seule pour le programme 6809, cette vue permet à l'utilisateur de vérifier et visualiser le code en binaire chargées au démarrage.

### 3. Modification des Registres

Comme vu dans la **Figure 1** , les champs de texte des registres ne sont pas de simples affichages. L'utilisateur peut cliquer sur une valeur (par exemple dans l'accumulateur A) et saisir une nouvelle valeur hexadécimale manuellement. Cela permet de tester des instructions sans avoir à modifier le code assembleur.

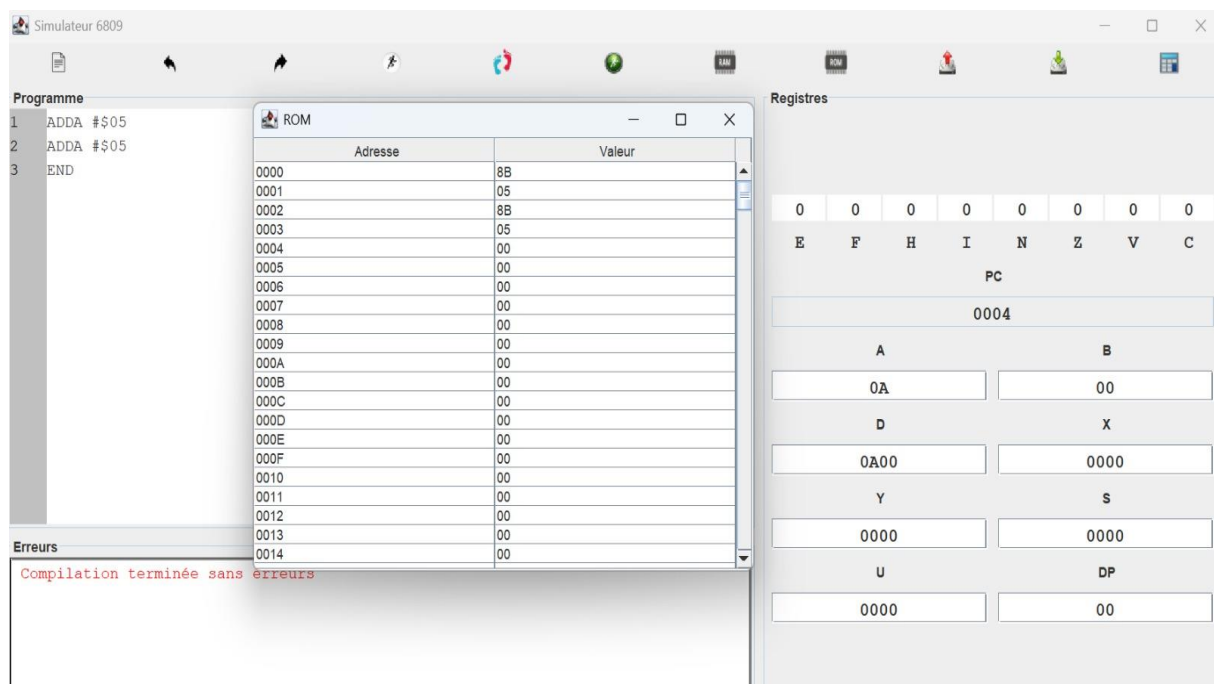
# Tests et validation

Pour garantir la fiabilité du simulateur, deux phases de tests ont été réalisées

## 1. Tests Unitaires

Chaque instruction a été testée isolément. Par exemple, pour l’instruction ADDA :

1. Chargement de 05 dans A.
2. Ajout de 05.
3. Vérification que A contient 0A.
4. Vérification des drapeaux (Flags N, Z, V, C).



## 2. Tests d'Intégration

Des programmes simples ont été exécutés, tels que le programme suivant, confirmant que la gestion de la mémoire (RAM) et des sauts (PC) fonctionne correctement.

```
; --- Chargement et Arithmétique 8-bits ---  
LDA #$10      ; Charger 16 (hex $10) dans A  
LDB #$05      ; Charger 05 dans B  
ADDA #$02     ; A = $10 + $02 = $12  
ADCB #$01     ; B = $05 + $01 + Carry (si présente)  
; --- Chargement et Arithmétique 16-bits ---  
LDD #$1000    ; Charger $1000 dans le registre double D (A=$10, B=$00)  
ADDD #$0050   ; D = $1000 + $0050 = $1050  
  
; --- Gestion des Registres d'Index et Pointeurs ---  
LDX #$2000    ; Initialiser le registre d'index X à l'adresse $2000  
LDY #$3000    ; Initialiser Y à l'adresse $3000  
LDS #$01FF    ; Initialiser la pile système S  
LDU #$02FF    ; Initialiser la pile utilisateur U  
; --- Stockage en Mémoire (RAM) ---  
STA $2000     ; Stocker le contenu de A à l'adresse RAM $2000
```

# Conclusion

Ce projet a abouti à la création d'un simulateur de Motorola 6809 fonctionnel et visuel. L'interface graphique, détaillée dans ce rapport, offre un contrôle total sur l'exécution du code et une visibilité claire sur l'état interne de la machine (RAM, ROM, Registres).

## 1. Limites

Le simulateur ne gère pas encore les périphériques externes (clavier, écran simulé) ni les interruptions matérielles complexes ainsi que les étiquettes et les instructions branchements.

## 2. Améliorations possibles

L'ajout d'une coloration syntaxique dans la zone d'édition et la simulation d'un écran graphique simple seraient des évolutions pertinentes pour rendre l'outil encore plus pédagogique.