Université Hassan Premier de Settat
Faculté des Sciences et Techniques de Settat (FST Settat)
Département Maths informatique
Filière : Génie Informatique

Guide d'Utilisation

Diplôme Licence Sciences et Techniques

Sous le Thème

Réalisation d'un simulateur de microprocesseur Motorola 6809

Elaboré par :

RADI Salma

MOUKRIM Aya

FARAJI Safae

BENBOUCETTA Imane

Encadré par :

Hicham BENALLA

Soutenu le 26 Décembre 2025 devant le jury composé de :

Pr.	Examinateur, Faculté des Sciences et Techniques - Settat
Pr. Hicham BENALLA	Encadrant, Faculté des Sciences et Techniques - Settat

Année Universitaire : 2025-2026

Table des matières :

Table de figures :

1.Introduction

1.1 Qu'est-ce que le Simulateur Motorola 6809?

Le Simulateur Motorola 6809 est un environnement d'émulation complet qui permet de:

- Développer des programmes en langage assembleur 6809
- Tester et déboguer du code sans matériel physique
- Apprendre l'architecture des microprocesseurs
- Visualiser l'exécution pas à pas des instructions

1.2 Fonctionnalités Principales

✓ Émulation complète du jeu d'instructions Motorola 6809
✓ Éditeur de code intégré avec syntaxe assembleur
✓ Débogueur avec points d'arrêt et exécution pas à pas
✓ Visualisation temps réel des registres et de la mémoire
✓ Interface graphique intuitive et conviviale
✓ Exemples prédéfinis pour démarrer rapidement

1.3 Public Cible

- Étudiants en informatique et électronique
- Développeurs travaillant sur des systèmes embarqués
- Enseignants en architecture des ordinateurs
- Passionnés de rétro-informatique

2. Installation et Démarrage

2.1 Prérequis Système

Configuration minimale:

- Java Runtime Environment (JRE) 8 ou supérieur
- 2 GB de RAM
- 100 MB d'espace disque
- Résolution d'écran: 1280x720 minimum

Configuration recommandée:

- Java Runtime Environment (JRE) 11 ou supérieur

- 4 GB de RAM
- Résolution d'écran: 1920x1080

2.2 Lancement du Simulateur

Méthode 1: Exécution directe

java -jar Motorola6809Simulator.jar

Méthode 2: Via la classe principale

java -cp . motorola6809.ui.MainApp

Méthode 3: Via le lanceur

java -cp . motorola6809.ui.Launcher

2.3 Premier Démarrage

Au premier lancement, vous verrez apparaître:

1. Le Dashboard principal - Interface centrale de contrôle
2. La fenêtre Architecture - Visualisation des registres
3. Les autres fenêtres sont fermées par défaut

3. Vue d'Ensemble de l'Interface

3.1 Dashboard Principal

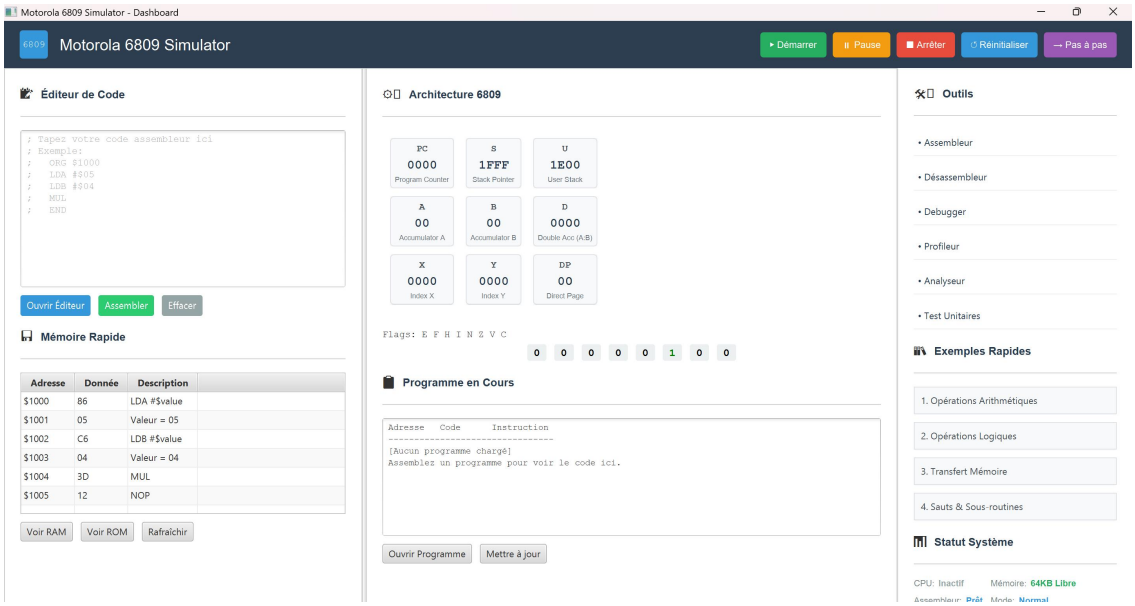


Figure 1 : Dashboard principale du simulateur

Le Dashboard est l'interface centrale qui intègre tous les composants du simulateur.

A. Barre Supérieure (Contrôles d'Exécution)

La barre supérieure contient les boutons de contrôle principaux:

▶ **Démarrer** (Vert)

- Lance l'exécution continue du programme
- Le programme s'exécute jusqu'à rencontrer un point d'arrêt ou une instruction SWI

⏸ **Pause** (Orange)

- Suspend l'exécution en cours
- Permet d'inspecter l'état du système
- Appuyez à nouveau pour reprendre

■ **Arrêter** (Rouge)

- Arrête complètement l'exécution
- Réinitialise le compteur de programme (PC)

↺ **Réinitialiser** (Bleu)

- Remet tous les registres à zéro
- Efface la mémoire
- Réinitialise le programme

→ **Pas à pas** (Violet)

- Exécute une seule instruction
- Idéal pour le débogage ligne par ligne
- Affiche l'état après chaque instruction

B. Panneau Gauche - Éditeur de Code

Zone de saisie:

- Tapez directement votre code assembleur
- Supporte la syntaxe 6809 standard
- Exemple de code fourni par défaut

Boutons:

- **Ouvrir Éditeur**: Lance l'éditeur dédié (fenêtre séparée)

- **Assembler**: Compile le code en langage machine
- **Effacer**: Vide la zone de texte

Section Mémoire Rapide:

- Tableau affichant les premières adresses mémoire
- Colonnes: Adresse | Donnée | Description
- Mise à jour automatique après assemblage

Boutons mémoire:

- **Voir RAM**: Ouvre la fenêtre RAM complète
- **Voir ROM**: Ouvre la fenêtre ROM
- **Rafraîchir**: Met à jour l'affichage mémoire

C. Panneau Central - Architecture 6809

Registres Généraux:

- **PC (Program Counter)**: Adresse de l'instruction en cours
- **S (Stack Pointer)**: Pointeur de pile système
- **U (User Stack)**: Pointeur de pile utilisateur

Accumulateurs:

- **A**: Accumulateur A (8 bits)
- **B**: Accumulateur B (8 bits)
- **D**: Double accumulateur (A:B, 16 bits)

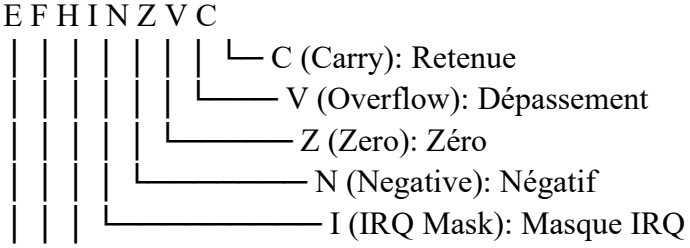
Registres d'Index:

- **X**: Registre d'index X (16 bits)
- **Y**: Registre d'index Y (16 bits)

Autres Registres:

- **DP (Direct Page)**: Registre de page directe (8 bits)

Flags (Condition Code Register):



```
│ │ └──────────────── H (Half-carry): Demi-retenue
│ └────────────────── F (FIRQ Mask): Masque FIRQ
└──────────────────── E (Entire): État complet sauvegardé
```

Programme en Cours:

- Affiche le code désassemblé
- Colonnes: Adresse | Code | Instruction
- Bouton **Ouvrir Programme**: Fenêtre dédiée
- Bouton **Mettre à jour**: Rafraîchit l'affichage

D. Panneau Droit - Outils

Section Outils: Liste des outils disponibles:

- **Assembleur**: Compile le code assembleur
- **Désassembleur**: Convertit le code machine en assembleur
- **Debugger**: Outils de débogage avancés
- **Profileur**: Analyse de performance
- **Analyseur**: Analyse statique du code
- **Test Unitaires**: Interface de tests

Section Exemples Rapides: Quatre exemples prédéfinis:

1. **Opérations Arithmétiques**: LDA, LDB, MUL, INCA, DECB
2. **Opérations Logiques**: COMA, COMB, EORA, EORB
3. **Transfert Mémoire**: STA, STB, LDX, STX
4. **Sauts & Sous-routines**: JMP, JSR, RTS, NOP

Section Statut Système:

- **CPU**: État (Actif/Inactif)
- **Mémoire**: Espace disponible (64KB)
- **Assembleur**: État (Prêt/En cours)
- **Mode**: Normal/Pause

Bouton Architecture Détail:

- Ouvre la fenêtre de visualisation détaillée des registres

4. Éditeur de Code

4.1 Fenêtre EDITEUR

Figure 2 : Fenêtre editeur de code

L'éditeur dédié offre plus d'espace pour écrire du code.

Composants de l'Éditeur :

Bouton "New"

- Efface tout le contenu
- Réinitialise les registres
- Prépare pour un nouveau programme

Bouton "Pas à Pas" :

- Compile le code (si nécessaire)
- Exécute une instruction
- Met à jour les registres
- Affiche le résultat dans la fenêtre Programme

Bouton "Exécuter" :

- Compile le code (si nécessaire)
- Lance l'exécution complète
- Continue jusqu'à SWI ou fin de programme
- Bascule en "Arrêter" pendant l'exécution

Zone de Texte :

- Police monospace pour meilleure lisibilité
- Support des commentaires (ligne commençant par ;)
- Taille: 250x230 pixels

4.2 Syntaxe du Code Assembleur

Structure d'un Programme

```
; Commentaire: Description du programme
        ORG $1400       ; Adresse de départ (obligatoire)

START:                  ; Étiquette (optionnelle)
        LDA #$05        ; Instruction avec commentaire
        LDB #$03
        MUL
        NOP
        END             ; Fin du programme (obligatoire)
```

Directives

ORG (Origin)

```
ORG $1000    ; Définit l'adresse de départ
ORG $C000    ; Peut être n'importe quelle adresse valide
```

END

```
END         ; Marque la fin du programme
```

Modes d'Adressage :

1. Immédiat (#)

```
LDA #$05     ; Charge la valeur 05 dans A
LDB #$FF     ; Charge la valeur FF dans B
LDD #$1234   ; Charge 1234 dans D (16 bits)
```

2. Direct ($)

```
LDA $50      ; Charge depuis l'adresse de page directe
STA $80      ; Stocke à l'adresse de page directe
```

3. Étendu

```
LDA $2000    ; Charge depuis l'adresse 2000
STA $3000    ; Stocke à l'adresse 3000
```

4. Indexé

```
LDA ,X       ; Charge depuis l'adresse pointée par X
STA ,Y       ; Stocke à l'adresse pointée par Y
```

5. Inhérent

```
NOP          ; Pas d'opérande
MUL          ; Utilise A et B automatiquement
INCA         ; Incrémente A
```

4.3 Instructions Disponibles

Arithmétiques

```
ADDA #$05    ; Addition (A = A + 5)
ADDB #$10    ; Addition (B = B + 16)
MUL          ; Multiplication (D = A * B)
INCA         ; Incrémentation (A = A + 1)
INCB         ; Incrémentation (B = B + 1)
DECA         ; Décrémentation (A = A - 1)
DECB         ; Décrémentation (B = B - 1)
DAA          ; Ajustement décimal après addition
```

Logiques

```
COMA         ; Complément de A (NOT A)
COMB         ; Complément de B (NOT B)
EORA #$FF    ; OU exclusif (A = A XOR FF)
EORB #$0F    ; OU exclusif (B = B XOR 0F)
```

Transfert

```
LDA #$value  ; Charger A
LDB #$value  ; Charger B
LDD #$value  ; Charger D (16-bit)
LDX #$addr   ; Charger X
LDY #$addr   ; Charger Y
LDS #$addr   ; Charger S
LDU #$addr   ; Charger U

STA $addr    ; Stocker A
```

```
STB $addr    ; Stocker B
STD $addr    ; Stocker D
STX $addr    ; Stocker X
STY $addr    ; Stocker Y
STS $addr    ; Stocker S
STU $addr    ; Stocker U
```

Contrôle de Flux

```
JMP $addr    ; Saut inconditionnel
JSR $addr    ; Appel de sous-routine
RTS          ; Retour de sous-routine
NOP          ; Aucune opération
CWAI #$FF    ; Attendre interruption
```

5. Architecture et Registres

5.1 Fenêtre "Architecture 6809 - Vue Détaillée"

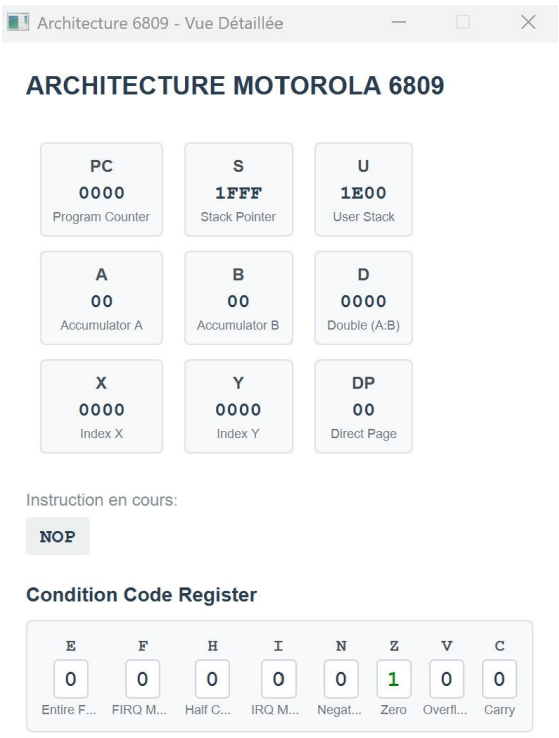


Figure 3 : Fenêtre Architrcture 6809 - Vue Detaillée

Cette fenêtre offre une visualisation complète et détaillée de tous les registres du processeur.

Affichage des Registres

Format d'Affichage:

- Nom du registre (ex: PC, A, B)
- Valeur en hexadécimal
- Description textuelle

Mise à Jour:

- Automatique après chaque instruction
- Temps réel pendant l'exécution pas à pas
- Les valeurs modifiées sont mises en évidence

Registres Principaux

PC (Program Counter) - 0000 à FFFF

- Pointe vers la prochaine instruction à exécuter
- Incrémenté automatiquement après chaque instruction
- Peut être modifié par les instructions de saut (JMP, JSR)

S (Stack Pointer) - Valeur initiale: 1FFF

- Pointeur de pile système
- Décrémenté lors des PUSH
- Incrémenté lors des PULL
- Utilisé pour les interruptions et JSR/RTS

U (User Stack Pointer) - Valeur initiale: 1E00

- Pointeur de pile utilisateur
- Séparé de la pile système
- Permet la gestion de contextes multiples

A (Accumulator A) - 8 bits

- Registre principal pour les opérations arithmétiques et logiques
- Partie haute de D

B (Accumulator B) - 8 bits

- Second accumulateur
- Partie basse de D

D (Double Accumulator) - 16 bits

- Combinaison de A:B

- Utilisé pour les opérations 16 bits
- Format: AABB (A=partie haute, B=partie basse)

X (Index Register X) - 16 bits

- Registre d'index principal
- Utilisé pour l'adressage indexé
- Peut être incrémenté/décrémenté

Y (Index Register Y) - 16 bits

- Second registre d'index
- Même fonctionnalité que X

DP (Direct Page) - 8 bits

- Définit la page mémoire pour l'adressage direct
- Permet un accès rapide à une page de 256 octets

5.2 Registre de Condition (CCR)

Le registre CC contient 8 flags qui reflètent l'état du processeur:

E - Entire Flag (0x80)

- **1**: État complet sauvegardé lors d'une interruption
- **0**: État partiel sauvegardé
- Utilisé pour les interruptions FIRQ (état partiel) vs IRQ/NMI (état complet)

F - FIRQ Mask (0x40)

- **1**: Interruptions FIRQ désactivées
- **0**: Interruptions FIRQ activées
- Contrôle les Fast Interrupts

H - Half Carry (0x20)

- **1**: Retenue du bit 3 au bit 4
- **0**: Pas de demi-retenue
- Utilisé pour l'arithmétique BCD (DAA)

I - IRQ Mask (0x10)

- **1**: Interruptions IRQ désactivées
- **0**: Interruptions IRQ activées
- Contrôle les interruptions normales

N - Negative (0x08)

- **1**: Résultat négatif (bit 7 = 1)
- **0**: Résultat positif (bit 7 = 0)
- Indique le signe du résultat

Z - Zero (0x04)

- **1**: Résultat égal à zéro
- **0**: Résultat différent de zéro
- Le flag le plus utilisé pour les tests

V - Overflow (0x02)

- **1**: Dépassement arithmétique signé
- **0**: Pas de dépassement
- Indique un résultat erroné en arithmétique signée

C - Carry (0x01)

- **1**: Retenue générée ou emprunt
- **0**: Pas de retenue
- Utilisé pour l'arithmétique multi-octets

Exemple de Lecture des Flags

État actuel: E F H I N Z V C
0 0 0 0 0 1 0 0

Signification:
- Z=1: Le dernier résultat était zéro
- Tous les autres flags sont à 0

6. Gestion de la Mémoire

6.1 Organisation de la Mémoire

Le Motorola 6809 dispose d'un espace d'adressage de 64 KB (0x0000 à 0xFFFF).

Carte Mémoire Standard :

0x0000 - 0x00FF : Page Directe (256 octets)
Accès rapide via adressage direct

0x0100 - 0x1DFF : RAM Utilisateur (7424 octets)

Zone libre pour programmes et données

0x1E00 - 0x1EFF : Pile Utilisateur (U) (256 octets)
Réservée pour la pile U

0x1F00 - 0x1FFF : Pile Système (S) (256 octets)
Réservée pour la pile S

0x2000 - 0x7FFF : RAM Étendue (~24 KB)
Données et programmes volumineux

0x8000 - 0xBFFF : Zone ROM/Programme (~16 KB)
Code programme principal

0xC000 - 0xFEFF : Périphériques I/O (~12 KB)
Mapped aux ports d'entrée/sortie

0xFF00 - 0xFFFF : Vecteurs d'Interruption (256 octets)
Adresses des routines d'interruption

6.2 Fenêtre RAM

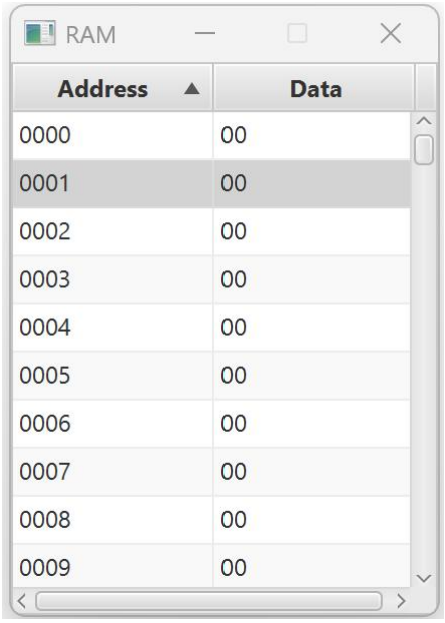


Figure 4 : Fenêtre RAM

La fenêtre RAM affiche la mémoire vive du système.

Caractéristiques

Format d'Affichage:

- **Address**: Adresse en hexadécimal (0000-03FF pour 1024 octets)
- **Data**: Valeur en hexadécimal (00-FF)

Taille:

- 1024 entrées visibles (0x0000 à 0x03FF)
- Défilement pour voir toutes les adresses
- Table redimensionnable

Initialisation:

- Toutes les valeurs à 00 au démarrage
- Modifiées par l'exécution du programme

Utilisation:

Visualisation:

- Ouvrir via "Voir RAM" dans le Dashboard
- Ou cliquer sur "Voir RAM" dans la section Mémoire Rapide

Lecture:

- Identifier l'adresse désirée
- Lire la valeur en hexadécimal
- Exemple: Adresse 2000 contient 05

Surveillance:

- Observer les changements pendant l'exécution
- Vérifier où le programme stocke ses données
- Déboguer les problèmes de mémoire

6.3 Fenêtre ROM

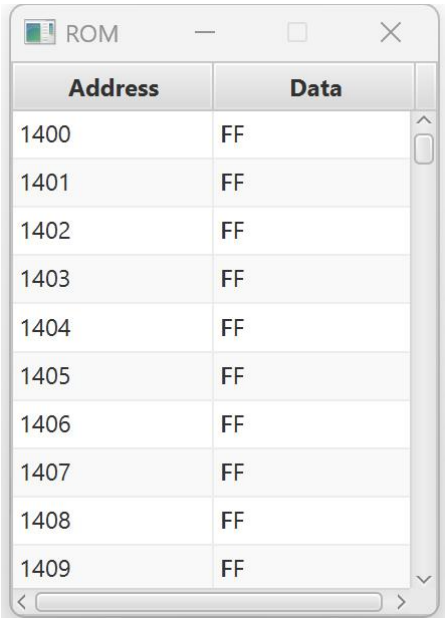


Figure 5 : Fenêtre ROM

La fenêtre ROM affiche la mémoire morte (Read-Only Memory).

Caractéristiques

Format d'Affichage:

- **Address**: Adresse en hexadécimal (1400-17FF typiquement)
- **Data**: Valeur en hexadécimal (FF par défaut)

Taille:

- 1024 entrées visibles (adresses 1400 à 17FF)
- Représente la zone ROM du système

Initialisation:

- Toutes les valeurs à FF au démarrage
- Remplies après assemblage d'un programme
- Lecture seule pendant l'exécution

Utilisation

Visualisation du Code Compilé: Après assemblage, vous verrez:

Address	Data
1400	86 (LDA #$05 - opcode)
1401	05 (valeur 05)
1402	C6 (LDB #$03 - opcode)
1403	03 (valeur 03)
1404	3D (MUL - opcode)
1405	12 (NOP - opcode)

Désassemblage Mental:

- Identifier les opcodes connus
- Retrouver les instructions originales
- Vérifier la compilation

6.4 Accès Mémoire dans le Code

Lecture

```
LDA $2000   ; Lit l'octet à l'adresse 2000 dans A
LDB $2001   ; Lit l'octet à l'adresse 2001 dans B
LDD $3000   ; Lit le mot (2 octets) à l'adresse 3000 dans D
LDX $4000   ; Lit le mot à l'adresse 4000 dans X
```

Écriture

```
STA $2000   ; Écrit A à l'adresse 2000
STB $2001   ; Écrit B à l'adresse 2001
STD $3000   ; Écrit D (2 octets) à l'adresse 3000
STX $4000   ; Écrit X (2 octets) à l'adresse 4000
```

7. Exécution de Programmes

7.1 Processus d'Exécution Complet

Étape 1: Écriture du Code

Dans l'éditeur:

```
ORG $1400
LDA #$05
LDB #$03
MUL
STA $2000
END
```

Étape 2: Assemblage

1. Cliquer sur **"Assembler"**
2. Le simulateur compile le code
3. Vérifie la syntaxe
4. Génère le code machine
5. Charge en mémoire ROM

Résultat attendu:

=== ASSEMBLAGE RÉUSSI ===
Taille: 6 octets

Addresse	Code	Instruction
1400	86 05	LDA #$05
1402	C6 03	LDB #$03
1404	3D	MUL
1405	B7 20 00	STA $2000

Étape 3: Vérification

- Consulter la fenêtre ROM pour voir le code compilé

- Vérifier les opcodes générés
- S'assurer que PC pointe vers 1400

Étape 4: Exécution

Option A: Exécution Complète

1. Cliquer sur **"▶ Démarrer"**
2. Le programme s'exécute entièrement
3. S'arrête sur SWI ou fin de programme
4. Les registres montrent l'état final

Option B: Exécution Pas à Pas

1. Cliquer sur **"→ Pas à pas"**
2. Une instruction s'exécute
3. Observer les changements de registres
4. Répéter pour l'instruction suivante

7.2 États d'Exécution

STOPPED (Arrêté)

- État initial du simulateur
- Aucun programme en cours
- Tous les contrôles disponibles
- PC = 0000

RUNNING (En cours)

- Programme en exécution continue
- Bouton "Démarrer" devient "Arrêter"
- Mise à jour automatique de l'affichage
- Peut être interrompu par Pause

PAUSED (En pause)

- Exécution temporairement suspendue
- État du système figé
- Possibilité d'inspecter les valeurs
- Reprise avec "Démarrer"

HALTED (Arrêté définitivement)

- Programme terminé (instruction SWI)
- Impossible de continuer

- Nécessite réinitialisation
- État final visible

7.3 Fenêtre Programme

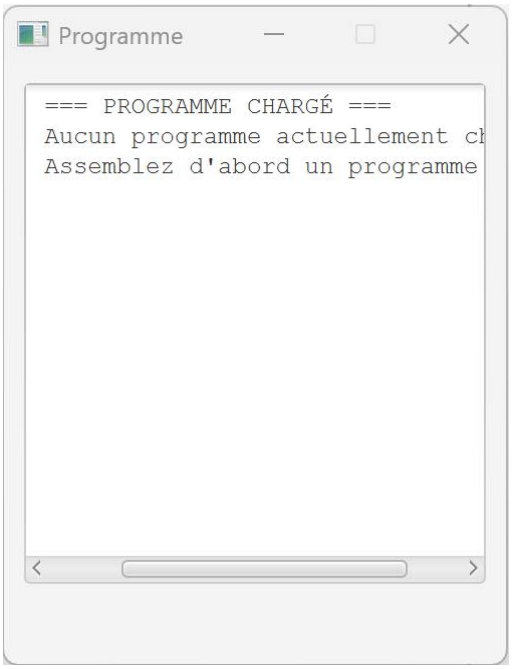


Figure 6 : Fenêtre programme

Affichage du Code en Exécution

Format:

Addresse	Code	Instruction
1400	86 05	LDA #$05
1402	C6 03	LDB #$03
1404	3D	MUL
1405	B7 20 00	STA $2000

Colonnes:

- **Adresse**: Position en mémoire

- **Code**: Octets en hexadécimal
- **Instruction**: Mnémonique assembleur

Utilisation:

- Suivre l'exécution du programme
- Identifier l'instruction courante (surlignée)
- Vérifier la séquence d'instructions

7.4 Suivi de l'Exécution

Instruction Courante

Dans la fenêtre Architecture:

Instruction en cours:
LDA #$05

Indique quelle instruction vient d'être exécutée.

Program Counter (PC)

- Pointe toujours vers la prochaine instruction
- Après "LDA #$05" à 1400, PC = 1402
- Permet de savoir où en est le programme

Exemple de Suivi Pas à Pas

État Initial:

PC = 1400, A = 00, B = 00
Instruction: LDA #$05

Après 1er pas:

PC = 1402, A = 05, B = 00
Instruction: LDB #$03

Après 2ème pas:

PC = 1404, A = 05, B = 03
Instruction: MUL

Après 3ème pas:

PC = 1405, A = 00, B = 0F, D = 000F (5*3=15=0F)

Instruction: STA $2000

Après 4ème pas:

PC = 1408, A = 00, Mémoire[2000] = 00
Programme terminé

8. Débogage

8.1 Exécution Pas à Pas

L'exécution pas à pas est l'outil principal de débogage.

Avantages

- **Contrôle total**: Une instruction à la fois
- **Observation**: Voir chaque changement
- **Compréhension**: Suivre la logique du programme
- **Détection d'erreurs**: Identifier où ça ne va pas

Procédure

Préparer le programme

✧ Assembler le code
✧ Vérifier qu'il n'y a pas d'erreurs

Positionner au début

✧ S'assurer que PC = adresse de départ
✧ Noter les valeurs initiales des registres

Exécuter pas à pas

✧ Cliquer sur "→ Pas à pas" ou "Pas à Pas" dans l'éditeur
✧ Observer les changements après chaque clic
✧ Noter les valeurs importantes

Analyser

✧ Les registres ont-ils les bonnes valeurs?
✧ La mémoire est-elle modifiée correctement?
✧ Le PC avance-t-il comme prévu?

8.2 Analyse des Registres

Vérifications Importantes

Après une instruction LDA:

Avant: A = 00
Après: A = 05 ✓

Après une instruction MUL:

Avant: A = 05, B = 03
Après: D = 000F (15 en décimal) ✓
A = 00, B = 0F

Après une instruction STA:

Mémoire avant: [2000] = 00
Mémoire après: [2000] = valeur de A ✓

8.3 Vérification de la Mémoire

Contrôle des Écritures

Noter l'adresse de destination

STA $2000 ; Va écrire à 2000

Ouvrir la fenêtre RAM

- Chercher l'adresse 2000
- Noter la valeur avant exécution

Exécuter l'instruction

- Un pas ou exécution complète

Vérifier le résultat

- La valeur a-t-elle changé?
- Est-ce la bonne valeur?

Détection de Corruptions

Symptômes:

- Valeurs inattendues en mémoire
- Programme qui plante

- Résultats incorrects

Causes possibles:

- Mauvaise adresse de destination
- Dépassement de pile
- Adressage incorrect

Solution:

- Exécution pas à pas pour isoler l'erreur
- Vérifier les adresses utilisées
- Contrôler les limites de pile

8.4 Analyse des Flags

Les flags donnent des informations cruciales sur l'état du système.

Flag Z (Zero)

Test:

```
LDA #$00     ; Charge 0
             ; Z devrait être 1
```

Vérification:

- Z = 1 si résultat = 0
- Z = 0 si résultat ≠ 0

Flag N (Negative)

Test:

```
LDA #$FF     ; Charge FF (négatif en signé)
             ; N devrait être 1
```

Vérification:

- N = 1 si bit 7 = 1
- N = 0 si bit 7 = 0

Flag C (Carry)

Test:

```
LDA #$FF
ADDA #$01    ; FF + 01 = 100 (dépassement)
             ; C devrait être 1
```

Vérification:

- C = 1 si retenue générée
- C = 0 sinon

Flag V (Overflow)

Test:

```
LDA #$7F     ; 127 (max positif en signé)
ADDA #$01    ; +1 = 128 = -128 (overflow!)
             ; V devrait être 1
```

Vérification:

- V = 1 si dépassement en arithmétique signée
- V = 0 sinon

8.5 Erreurs Courantes et Solutions

Erreur: Programme ne démarre pas

Symptômes:

- Rien ne se passe après "Démarrer"
- PC reste à 0000

Causes:

- Code non assemblé
- Directive ORG manquante
- Erreur de compilation

Solutions:

1. Vérifier que le code compile sans erreur
2. S'assurer que ORG est présent
3. Regarder les messages d'erreur

Erreur: Résultat incorrect

Symptômes:

- Le calcul donne un mauvais résultat
- Les registres ont des valeurs inattendues

Causes:

- Mauvais ordre des instructions
- Oubli d'une instruction
- Erreur de mode d'adressage

Solutions:

1. Exécution pas à pas pour identifier l'étape problématique
2. Vérifier la syntaxe de chaque instruction
3. Comparer avec l'exemple attendu

Erreur: Mémoire corrompue

Symptômes:

- Valeurs bizarres en mémoire
- Programme écrase ses propres données

Causes:

- Écriture à une mauvaise adresse
- Débordement de pile
- Calcul d'adresse erroné

Solutions:

1. Vérifier toutes les instructions STA/STB/STD
2. S'assurer que la pile ne déborde pas
3. Utiliser des adresses bien séparées pour code et données

9. Interfaces de Test

9.1 SimpleTestApp (Tester - Tous en un)

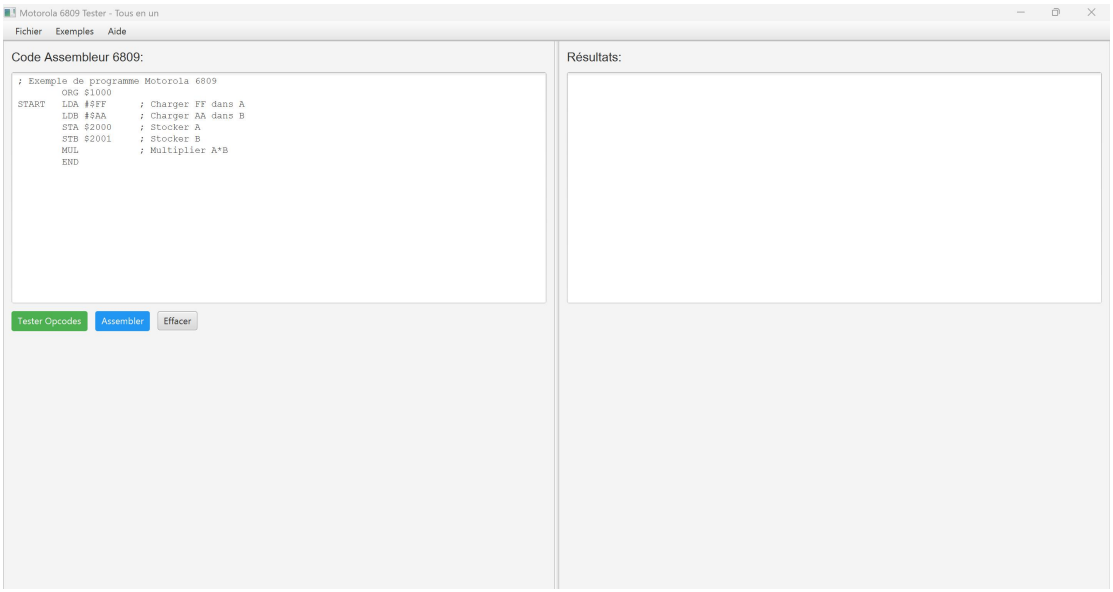


Figure 7 : Interface de Test

Interface complète pour tester et assembler du code.

Sections de l'Interface

A. Code Assembleur 6809 (Gauche)

Zone de saisie principale:

- Police monospace pour lisibilité
- Exemple de code par défaut
- Coloration basique

Boutons:

- **Tester Opcodes**: Lance une batterie de tests sur tous les opcodes
- **Assembler**: Compile le code et affiche les résultats
- **Effacer**: Vide la zone de code

B. Résultats (Droite)

Affiche:

- Résultats de l'assemblage
- Taille du code généré
- Code machine en hexadécimal
- Vue désassemblée
- Erreurs de compilation

C. Menu Supérieur

- **Fichier** → Quitter
- **Exemples** → 4 exemples prédéfinis
- **Aide** → À propos

Test des Opcodes

Cliquer sur **"Tester Opcodes"** lance une vérification complète:

Résultat typique:

=== TEST DE TOUS LES OPCODES ===

✓ COMA INHERENT : 0x43
✓ COMB INHERENT : 0x53
✓ INCA INHERENT : 0x4C
✓ INCB INHERENT : 0x5C
✓ DECA INHERENT : 0x4A
✓ DECB INHERENT : 0x5A
✓ NOP INHERENT : 0x12
✓ MUL INHERENT : 0x3D
✓ LDA IMMEDIATE : 0x86
✓ LDB IMMEDIATE : 0xC6
✓ EORA IMMEDIATE : 0x88

Résultat: 11/11 tests réussis

9.2 TestInterface

Interface avancée avec visualisation mémoire et registres.

Organisation de l'Interface

1. Panneau Gauche (Code Source)

- Zone de saisie de code assembleur
- 400 pixels de large
- Boutons: Assembler, Effacer, Exemple

Exemples rapides:

1. Opérations arithmétiques
2. Opérations logiques
3. Chargement/Stockage
4. Sauts et sous-routines

2. Panneau Central (Sortie et Registres)

Section Sortie:

- Résultats de l'assemblage
- Messages d'erreur
- Code désassemblé
- Statistiques

Section Registres:

PC: 0000 (Program Counter)
A: 00 (Accumulator A)
B: 00 (Accumulator B)
D: 0000 (A:B 16-bit)
X: 0000 (Index Register X)
Y: 0000 (Index Register Y)
S: 1FFF (Stack Pointer)
U: 1E00 (User Stack Pointer)
DP: 00 (Direct Page)

CCR (Condition Code Register):
E F H I N Z V C
0 0 0 0 0 1 0 0

3. Panneau Droit (Mémoire et Instructions)

Tableau Mémoire:

- Colonnes: Adresse | Donnée | ASCII | Description
- 16 premières adresses affichées
- Mise à jour après assemblage
- Interprétation ASCII des données

Instructions Disponibles: Liste complète organisée par catégorie:

=== ARITHMÉTIQUES ===
INCA, INCB, INC : Incrémentation
DECA, DECB, DEC : Décrémentation
ADDA, ADDB : Addition
MUL : Multiplication
DAA : Ajustement décimal

=== LOGIQUES ===
COMA, COMB, COM : Complément

EORA, EORB : OU exclusif

=== TRANSFERT ===
LDA, LDB, LDD : Chargement
STA, STB, STD : Stockage
LDX, LDY, LDS, LDU : Chargement index
STX, STY, STS, STU : Stockage index
EXG : Échange

=== SAUT ===
JMP, JSR : Saut
RTS : Retour sous-routine

=== CONTRÔLE ===
NOP : Aucune opération
CWAI : Attente interruption

Utilisation de TestInterface

Flux de Travail:

Sélectionner un exemple

- ✓ Cliquer sur un des 4 boutons d'exemple
- ✓ Le code est chargé automatiquement

Assembler

- ✓ Cliquer sur "Assembler"
- ✓ Observer les résultats dans la section Sortie

Analyser

- ✓ Vérifier le code machine généré
- ✓ Examiner les registres simulés
- ✓ Consulter la mémoire

Modifier et réessayer

- ✓ Modifier le code selon les besoins
- ✓ Effacer avec "Effacer" si nécessaire
- ✓ Réassembler

10. Exemples Pratiques

10.1 Exemple : Opérations Arithmétiques

Code Source

```
; === OPÉRATIONS ARITHMÉTIQUES ===
        ORG $1000

        LDA #$09        ; A = 09
        INCA            ; A = 0A (+1)

        LDB #$10        ; B = 10
        DECB            ; B = 0F (-1)

        LDA #$05        ; A = 05
        LDB #$04        ; B = 04
        MUL             ; D = 0014 (5*4=20)

        LDA #$99        ; Test DAA
        ADDA #$01       ; A = 9A
        DAA             ; Correction BCD -> A = 00, C = 1

        END
```

Analyse Ligne par Ligne

Ligne 1-2: LDA #$09, INCA

Avant: A = 00
Après: A = 09 puis A = 0A
Flags: N=0, Z=0, V=0, C=0

Ligne 3-4: LDB #$10, DECB

Avant: B = 00
Après: B = 10 puis B = 0F
Flags: N=0, Z=0, V=0, C=0

Ligne 5-7: LDA #$05, LDB #$04, MUL

Avant: A = 0A, B = 0F
Après: A = 05, B = 04
Après MUL: D = 0014 (20 en décimal)
A = 00, B = 14
Flags: Z=0, C=0

Ligne 8-10: Test DAA

LDA #$99: A = 99
ADDA #$01: A = 9A (99+1 en hexadécimal)
DAA: Ajustement BCD
A = 00 (car 99+1=100 en BCD)
C = 1 (retenue)

Résultat Attendu

- A = 00
- B = 14
- D = 0014
- Flags C = 1, Z = 1

11.Conclusion

Le Motorola 6809, bien que conçu dans les années 1970, reste un processeur remarquable par son architecture avancée et son jeu d'instructions riche. Ce simulateur honore cet héritage en le rendant accessible aux nouvelles générations de développeurs et d'étudiants.

En offrant une fenêtre sur le passé tout en utilisant des technologies modernes, ce projet illustre parfaitement comment la connaissance historique peut éclairer l'innovation future. Il démontre que l'étude des systèmes anciens n'est pas qu'une curiosité académique, mais une source précieuse d'inspiration et de compréhension pour les défis actuels de l'informatique.

Que vous soyez étudiant découvrant l'assembleur pour la première fois, développeur travaillant sur des systèmes embarqués, ou simplement passionné par l'histoire de l'informatique, nous espérons que ce simulateur vous sera utile et inspirant.