Simulateur de Microprocesseur Motorola 6809

Documentation Technique Détaillée et Guide d'Utilisation

---

1. Introduction Générale

Le microprocesseur Motorola 6809, apparu à la fin des années 1970, est reconnu comme l'un des processeurs 8 bits les plus avancés de son époque. Il se distingue par une architecture orthogonale, un jeu d'instructions riche et une séparation claire entre données et adresses.

Ce projet vise à développer un simulateur logiciel du Motorola 6809, écrit en Java, permettant :

- L'apprentissage du langage assembleur 6809
- La compréhension du fonctionnement interne du processeur
- Le test et le débogage de programmes assembleur
- La visualisation graphique de l'état du processeur et de la mémoire

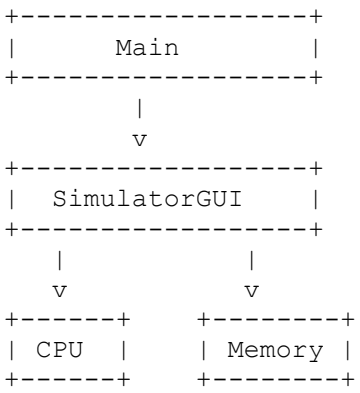
Le simulateur intègre une émulation du CPU, un assembleur simplifié, un débogueur pas à pas et une interface graphique interactive.

---

2. Architecture Globale du Système

Le simulateur repose sur une architecture modulaire, facilitant la compréhension et l'évolution du projet.

2.1 Diagramme Conceptuel



2.2 Description des Modules

Module	Rôle
Main	Démarrage de l'application
SimulatorGUI	Interface graphique, assembleur, débogueur
Cpu6809	Émulation du processeur
Memory	Gestion mémoire RAM / ROM

---

3. Émulation du Processeur – Classe Cpu6809

3.1 Rôle et Responsabilités

La classe Cpu6809 représente le cœur du simulateur.

Elle est responsable de :

- La gestion des registres internes
- Le décodage des instructions
- L'exécution instruction par instruction
- La mise à jour des drapeaux
- Le suivi de l'instruction courante

3.2 Registres du Processeur

Registres 16 bits

Registre	Description
PC	Compteur ordinal
S	Pile système
U	Pile utilisateur
X	Registre index
Y	Registre index

Registres 8 bits

Registre	Description
A	Accumulateur A
B	Accumulateur B
DP	Direct Page
CC	Code Condition

3.3 Code Condition (CC)

Seuls les drapeaux suivants sont implémentés :

Bit	Nom	Rôle
N	Négatif	Résultat signé négatif
Z	Zéro	Résultat nul

Les autres drapeaux sont présents visuellement mais non fonctionnels.

3.4 Initialisation du Processeur

La méthode reset() :

- Initialise le PC à 0xF000
- Réinitialise tous les registres
- Efface les drapeaux
- Indique que le programme n’est pas terminé

---

4. Cycle d’Exécution du CPU

4.1 Principe Général

Chaque appel à la méthode :

```
cpu.step();
```

représente un cycle processeur.

4.2 Algorithme du Cycle

1. Lire l’opcode à l’adresse PC
2. Incrémenter le PC
3. Décoder l’opcode via un switch
4. Lire les opérandes si nécessaire
5. Exécuter l’instruction
6. Mettre à jour les drapeaux
7. Mettre à jour l’instruction courante

4.3 Exemple : Instruction LDA #imm

```
LDA #$10
```

Étapes internes :

- Lecture opcode 0x86
- Lecture immédiate de la valeur 0x10
- Chargement dans A
- Mise à jour du drapeau Z et N

---

5. Jeu d’Instructions Implémenté

5.1 Instructions de Chargement

- LDA, LDB
- LDX, LDS

5.2 Instructions de Stockage

- STA, STB

5.3 Instructions Arithmétiques

- ADDA, ADDB
- INCA, INCB
- DECA, DECB

5.4 Instructions de Comparaison

- CMPX

5.5 Instructions de Branchement

- BRA (inconditionnel)
- BEQ (si Z = 1)
- BNE (si Z = 0)
- JMP

5.6 Instructions Spéciales

- EXG A,DP
- TFR A,DP
- PSHS, PULS
- CLRA, CLRB
- NOP
- END / SWI

---

6. Gestion de la Mémoire – Classe Memory

6.1 Organisation Mémoire

Plage	Type	Initialisation
0000–7FFF	RAM	00
8000–FFFF	ROM	FF

6.2 Accès Mémoire

- Lecture toujours autorisée
- Écriture autorisée sur toute la mémoire (simplification pédagogique)

---

7. Assembleur Intégré

7.1 Fonctionnement Général

L’assembleur intégré traduit un langage assembleur simplifié directement en opcodes machine.

7.2 Algorithme d’Assemblage

1. Lecture ligne par ligne
2. Suppression des espaces
3. Conversion en majuscules
4. Validation syntaxique stricte
5. Identification du mnémotechnique
6. Détermination du mode d’adressage
7. Écriture de l’opcode
8. Écriture des opérandes
9. Chargement en mémoire à partir de 0xF000

7.3 Validation Syntaxique

Exemples d’erreurs détectées :

- Absence de # pour immédiat
- Mauvais symbole ($#, #<)
- Valeur hexadécimale invalide

- Mnémotechnique inconnu

---

8. Interface Graphique et Débogueur

8.1 Fenêtres Principales

- Architecture interne
- Éditeur assembleur
- Mémoire RAM
- Mémoire ROM
- Désassemblage du programme

8.2 Débogueur Pas à Pas

Fonctionnalités :

- RESET du processeur
- Exécution instruction par instruction
- Mise à jour automatique :
    - Registres
    - Mémoire
    - Instruction courante

---

9. Conclusion Générale

Ce simulateur du Motorola 6809 constitue une plateforme pédagogique robuste, permettant :

- La compréhension de l'architecture interne
- L'apprentissage du langage assembleur
- Le débogage précis pas à pas
- Une visualisation claire et intuitive du fonctionnement du processeur

Il offre une base solide pour des extensions futures et des projets académiques avancés.

---

12. Analyse Détaillée Instruction par Instruction

Ce chapitre décrit précisément le fonctionnement interne de chaque instruction implémentée, depuis la lecture de l'opcode jusqu'à la mise à jour des registres et des drapeaux.

12.1 Instructions de Chargement (LOAD)

12.1.1 LDA – Load Accumulator A

a) Mode Immédiat

```
LDA #$NN
```

Opcode : 0x86

Taille : 2 octets

Fonctionnement interne :

1. Lecture de l'opcode à l'adresse PC
2. Incrémentation du PC
3. Lecture de l'octet immédiat
4. Chargement dans le registre A
5. Mise à jour des drapeaux N et Z

Code Java :

```
case 0x86:
    A = memory.read(PC++);
    updateFlags(A);
```

---

b) Mode Direct

```
LDA <$NN
```

Opcode : 0x96

Adresse effective :

```
EA = (DP << 8) | NN
```

Fonctionnement :

- Lecture de l'octet en mémoire directe
- Chargement dans A
- Mise à jour des drapeaux

---

c) Mode Indexé

```
LDA ,X
```

Opcode : 0xA6

- L'adresse effective correspond au contenu du registre X
- L'octet pointé est chargé dans A

12.1.2 LDB – Load Accumulator B

Même logique que LDA, mais avec le registre B.

Mode	Opcode
Immédiat	0xC6
Direct	0xD6
Indexé	0xE6

12.2 Instructions de Stockage (STORE)

12.2.1 STA – Store Accumulator A

```
STA <$NN
```

Opcode : 0x97

Fonctionnement :

- Calcul de l'adresse effective
- Écriture du contenu de A en mémoire
- Mise à jour des drapeaux selon la valeur écrite

12.2.2 STB – Store Accumulator B

Même fonctionnement que STA, mais avec le registre B.

12.3 Instructions Arithmétiques

12.3.1 ADDA – Addition Accumulateur A

```
ADDA #$NN
```

Opcode : 0x8B

Fonctionnement :

1. Lecture de l'opérande immédiat
2. Addition avec A
3. Résultat masqué sur 8 bits
4. Mise à jour des drapeaux N et Z

```
A = (A + value) & 0xFF;
updateFlags(A);
```

12.3.2 ADDB – Addition Accumulateur B

Même logique que ADDA, appliquée à B.

12.3.3 INCA / INCB

```
INCA
```

Opcode : 0x4C

- Incrémente le registre de 1
- Mise à jour des drapeaux

12.3.4 DECA / DECB

```
DECA
```

Opcode : 0x4A

- Décrémente le registre
- Résultat limité à 8 bits
- Mise à jour des drapeaux

12.4 Instructions 16 bits

12.4.1 LDX – Load Index Register X

```
LDX #$HHLL
```

Opcode : 0x8E

Taille : 3 octets

Fonctionnement :

- Lecture des deux octets (High puis Low)
- Assemblage sur 16 bits
- Chargement dans X
- Mise à jour du drapeau Z uniquement

12.4.2 LDS – Load Stack Pointer S

Même logique que LDX, appliquée au registre S.

12.4.3 CMPX – Compare X

```
CMPX #$HHLL
```

Opcode : 0x8C

Fonctionnement :

- Soustraction logique X - valeur
- Résultat non stocké
- Drapeau Z mis à 1 si égalité

12.5 Instructions de Branchement

12.5.1 BRA – Branch Always

```
BRA offset
```

Opcode : 0x20

- L'offset est signé sur 8 bits
- Le PC est modifié relativement

```
PC += (byte)offset;
```

12.5.2 BEQ – Branch if Equal

```
BEQ offset
```

Opcode : 0x27

- Saut exécuté si Z = 1

12.5.3 BNE – Branch if Not Equal

```
BNE offset
```

Opcode : 0x26

- Saut exécuté si Z = 0

12.6 Instructions de Transfert et Échange

12.6.1 EXG A,DP

```
EXG A,DP
```

Opcode : 0x1E 0x8B

- Échange du contenu de A et DP
- Aucun drapeau modifié

12.6.2 TFR A,DP

```
TFR A,DP
```

Opcode : 0x1F 0x8B

- Copie de A vers DP

12.7 Instructions de Pile

12.7.1 PSHS

```
PSHS
```

Opcode : 0x34

- Décrémente S
- Simule l’empilement (simplifié)

12.7.2 PULS

```
PULS
```

Opcode : 0x35

- Incrémente S
- Simule le dépilement

12.8 Instructions de Contrôle

12.8.1 JMP – Saut Absolu

```
JMP $HHLL
```

Opcode : 0x7E

- Charge directement le PC

12.8.2 NOP

```
NOP
```

Opcode : 0x12

- Aucun effet

12.8.3 CLRA / CLRB

```
CLRA
```

- Met le registre à zéro
- Drapeau Z activé

12.8.4 END / SWI

```
END
```

Opcode : 0x3F

- Arrête l’exécution
- Marque la fin du programme