Université Hassan Premier de Settat
Faculté des Sciences et Techniques de Settat

RAPPORT DE PROJET

Réalisation d'un Simulateur de
Microprocesseur Motorola 6809

Filière: Génie Informatique

Encadré par: Pr. Hicham BENALLA

Année Universitaire: 2025-2026

Date de soutenance: 26 Décembre 2025

Table des Matières

1. Introduction

Le Simulateur Motorola 6809 est un environnement d'émulation complet qui permet de développer, tester et déboguer des programmes en langage assembleur pour le microprocesseur Motorola 6809. Ce projet vise à fournir une plateforme pédagogique et professionnelle pour l'apprentissage de l'architecture des microprocesseurs et le développement de systèmes embarqués.

1.1 Qu'est-ce que le Motorola 6809?

Le Motorola 6809 est un microprocesseur 8 bits conçu dans les années 1970, reconnu pour son architecture avancée et son jeu d'instructions riche. Malgré son âge, il reste une excellente référence pour l'apprentissage des concepts fondamentaux de l'architecture des ordinateurs.

1.2 Public Cible

Ce simulateur s'adresse aux étudiants en informatique et électronique, aux développeurs travaillant sur des systèmes embarqués, aux enseignants en architecture des ordinateurs, et aux passionnés de rétro-informatique.

2. Objectifs du Projet

Objectif	Description
Émulation complète	Implémenter l'ensemble du jeu d'instructions du 6809
Débogueur intégré	Points d'arrêt, exécution pas à pas, visualisation mémoire
Interface graphique	Interface utilisateur intuitive et conviviale
Environnement d'apprentissage	Faciliter l'apprentissage de l'assembleur

3. Fonctionnalités Principales

3.1 Environnement d'Émulation

✓ Émulation complète du jeu d'instructions Motorola 6809
✓ Prise en charge de tous les modes d'adressage
✓ Suivi en temps réel des registres du processeur
✓ Capacité à définir et modifier manuellement l'état du processeur

3.2 Débogueur Intégré

✓ Définition de points d'arrêt pour suspendre l'exécution
✓ Exécution pas à pas avec affichage détaillé
✓ Inspection et modification de la mémoire pendant l'exécution
✓ Visualisation graphique de la mémoire (RAM et ROM)

3.3 Interface Utilisateur

✓ Console principale intégrant tous les composants
✓ Éditeur de code assembleur avec coloration syntaxique
✓ Fenêtres d'architecture affichant l'état des registres
✓ Visualisation des flags en temps réel
✓ Vues mémoire séparées pour RAM et ROM

4. Architecture et Conception

4.1 Architecture Générale

Le simulateur est basé sur une architecture modulaire en Java, utilisant le framework Spring Boot pour la gestion des composants. L'architecture se compose de trois couches principales : la couche présentation (UI), la couche logique (Backend), et la couche matérielle (Core).

4.2 Composants Principaux

• CPU (Unité Centrale de Traitement)
Émule le microprocesseur 6809 avec ses registres, flags et jeu d'instructions.

• Memory (Mémoire)
Gère la RAM (64 KB) avec zones distinctes pour code et données.

• Assembler (Assembleur)
Convertit le code assembleur en code machine exécutable.

• Debugger (Débogueur)
Fournit les outils de débogage (points d'arrêt, exécution pas à pas).

5. Algorithmes de Configuration

5.1 Algorithme d'Initialisation du CPU

ALGORITHME: InitialiserCPU()
ENTRÉE: Aucune
SORTIE: CPU initialisé à l'état par défaut

DÉBUT
  // Initialisation des registres 8 bits
  A ← 0x00
  B ← 0x00
  DP ← 0x00

  // Initialisation des registres 16 bits
  X ← 0x0000
  Y ← 0x0000
  U ← 0x0000
  S ← 0x0000
  PC ← 0x1400 // Adresse de départ en ROM

  // Initialisation des flags
  E ← 0 // Entire state not saved
  F ← 0 // FIRQ not masked
  H ← 0 // Half carry clear
  I ← 0 // IRQ not masked
  N ← 0 // Not negative
  Z ← 1 // Zero (état initial)
  V ← 0 // No overflow
  C ← 0 // No carry

  // Lecture du vecteur de RESET
  SI Mémoire[0xFFFE:0xFFFF] ≠ 0xFFFF ALORS
    PC ← Mémoire[0xFFFE] << 8 | Mémoire[0xFFFF]
  FIN SI
FIN

5.2 Algorithme d'Assemblage (2 Passes)

ALGORITHME: Assembler(code_source)
ENTRÉE: code_source (String) - Code assembleur
SORTIE: code_machine (Array) - Code binaire

DÉBUT
  symbolTable ← TableDesSymboles()
  adresse_courante ← 0x1400
  code_machine ← []

  // PREMIÈRE PASSE: Construction table symboles
  lignes ← DécouperEnLignes(code_source)
  POUR CHAQUE ligne DANS lignes FAIRE
    SI EstLigneVide(ligne) OU EstCommentaire(ligne) ALORS
      CONTINUER
    FIN SI

    (étiquette, mnémonique, opérande) ← ParserLigne(ligne)

    // Traitement directive ORG
    SI mnémonique = "ORG" ALORS
      adresse_courante ← ConvertirEnHex(opérande)
      CONTINUER
    FIN SI

    // Enregistrement étiquette
    SI étiquette ≠ NULL ALORS
      symbolTable.AjouterÉtiquette(étiquette, adresse_courante)
    FIN SI

    taille ← CalculerTailleInstruction(mnémonique, opérande)
    adresse_courante ← adresse_courante + taille
  FIN POUR

  // DEUXIÈME PASSE: Génération code machine
  adresse_courante ← 0x1400
  POUR CHAQUE ligne DANS lignes FAIRE
    (étiquette, mnémonique, opérande) ← ParserLigne(ligne)

    SI mnémonique = "END" ALORS
      SORTIR DE LA BOUCLE
    FIN SI

    mode_adressage ← DéterminerModeAdressage(opérande)
    opcode ← ObtenirOpcode(mnémonique, mode_adressage)
    code_machine.Ajouter(opcode)

    SI opérande ≠ NULL ALORS
      octets_opérande ← EncoderOpérande(opérande, mode_adressage)
      code_machine.Ajouter(octets_opérande)

```
    FIN SI
  FIN POUR

  RETOURNER code_machine
FIN
```

5.3 Cycle d'Exécution (Fetch-Decode-Execute)

ALGORITHME: CycleExécution()
ENTRÉE: Aucune
SORTIE: nombre_cycles (Integer)

DÉBUT
  SI halted = VRAI ALORS
    RETOURNER 0
  FIN SI

  // PHASE FETCH: Lecture de l'instruction
  opcode ← Mémoire[PC]
  PC ← PC + 1

  // Gestion opcodes 2 octets
  SI opcode ∈ {0x10, 0x11} ALORS
    second_octet ← Mémoire[PC]
    PC ← PC + 1
    opcode ← (opcode << 8) | second_octet
  FIN SI

  // PHASE DECODE: Identification instruction
  instruction ← TableInstructions[opcode]
  SI instruction = NULL ALORS
    LEVER Erreur("Opcode inconnu")
  FIN SI

  // PHASE EXECUTE: Exécution instruction
  cycles ← instruction.Exécuter(CPU)
  cycleCount ← cycleCount + cycles

  RETOURNER cycles
FIN

5.4 Mise à Jour des Flags

ALGORITHME: MettreÀJourFlags(opération, opérande1, opérande2, résultat, taille_bits)

DÉBUT
  masque ← SI taille_bits = 8 ALORS 0xFF SINON 0xFFFF
  bit_signe ← SI taille_bits = 8 ALORS 0x80 SINON 0x8000
  résultat_masqué ← résultat ET masque

  // FLAG ZERO (Z)
  Z ← SI résultat_masqué = 0 ALORS 1 SINON 0

  // FLAG NEGATIVE (N)
  N ← SI (résultat_masqué ET bit_signe) ≠ 0 ALORS 1 SINON 0

  // FLAG CARRY (C) et OVERFLOW (V)
  SI opération = "ADD" ALORS
    // Carry: débordement non signé
    C ← SI résultat > masque ALORS 1 SINON 0

    // Overflow: débordement signé
    signe_op1 ← (opérande1 ET bit_signe) ≠ 0
    signe_op2 ← (opérande2 ET bit_signe) ≠ 0
    signe_res ← (résultat_masqué ET bit_signe) ≠ 0
    V ← SI (signe_op1 = signe_op2) ET
          (signe_op1 ≠ signe_res) ALORS 1 SINON 0

    // Half-Carry (pour addition 8 bits)
    SI taille_bits = 8 ALORS
      H ← SI ((opérande1 ET 0x0F) +
            (opérande2 ET 0x0F)) > 0x0F ALORS 1 SINON 0
    FIN SI
  FIN SI
FIN

6. Guide d'Utilisation

6.1 Installation et Démarrage

Prérequis Système:
• Java Runtime Environment (JRE) 8 ou supérieur
• 2 GB de RAM minimum (4 GB recommandé)
• 100 MB d'espace disque
• Résolution d'écran: 1280x720 minimum

Lancement du Simulateur:
java -jar Motorola6809Simulator.jar

Au premier lancement, la console principale s'affiche avec l'éditeur de code intégré.

6.2 Interface Utilisateur

Console Principale:
• Barre supérieure: État d'exécution et registres rapides
• Panneau gauche: Éditeur de code assembleur
• Panneau droit: Contrôles d'exécution et accès aux fenêtres
• Panneau inférieur: Console d'exécution et messages système

Boutons de Contrôle:
■ **Démarrer:** Lance l'exécution continue du programme
■ **Pause:** Suspend temporairement l'exécution
■ **Arrêter:** Arrête complètement l'exécution
→ **Pas à pas:** Exécute une seule instruction
■ **Réinitialiser:** Remet tous les registres à zéro

6.3 Écriture de 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
  STA $2000 ; Stocker résultat
  SWI ; Retour au système

END ; Fin du programme (obligatoire)
```

Règles de Formatage:
- Les commentaires commencent par ;
- Les étiquettes se terminent par :
- Les instructions sont indentées
- ORG définit l'adresse de départ
- END marque la fin du programme

6.4 Modes d'Adressage

1. Immédiat (#): La valeur est dans l'instruction
```
  LDA #$05 ; Charge la valeur 05 dans A
```

2. Direct ($): Adresse dans la page directe (0000-00FF)
```
  LDA $50 ; Charge depuis l'adresse $0050
```

3. Étendu: Adresse complète 16 bits
```
  LDA $2000 ; Charge depuis l'adresse $2000
```

4. Indexé: Utilise un registre d'index
```
  LDA ,X ; Charge depuis l'adresse pointée par X
  LDB 5,X ; Charge depuis X+5
```

5. Inhérent: Pas d'opérande
```
  NOP ; Aucune opération
  MUL ; Multiplier A × B → D
```

7. Exemples Pratiques

7.1 Exemple Simple: Addition

```
; Programme d'addition simple
ORG $1000

START:
  LDA #$05 ; A = 05
  LDB #$03 ; B = 03
  ADDA B ; A = A + B = 08
  STA $2000 ; Stocker à $2000
  SWI ; Fin

END
```

Résultat attendu:
• A = 08
• Mémoire[$2000] = 08
• Flags: Z=0, N=0, V=0, C=0

7.2 Exemple: Multiplication

```
; Programme de multiplication
ORG $1400

  LDA #$05 ; A = 5
  LDB #$04 ; B = 4
  MUL ; D = A × B = 20 (0x14)
  STD $3000 ; Stocker D à $3000
  SWI

END
```

Résultat attendu:
• D = 0014 (20 en décimal)
• A = 00, B = 14
• Mémoire[$3000] = 00
• Mémoire[$3001] = 14

7.3 Exemple: Boucle

```
; Programme avec boucle (somme 1 à 5)
ORG $1000

  LDA #$00 ; A = 0 (somme)
  LDB #$01 ; B = 1 (compteur)

LOOP:
  ADDA B ; A = A + B
  INCB ; B = B + 1
  CMPB #$06 ; Comparer B avec 6
  BNE LOOP ; Si B ≠ 6, continuer

  STA $2000 ; Stocker résultat
  SWI

END
```

Résultat attendu:
• A = 15 (0x0F) ; somme de 1+2+3+4+5
• B = 06
• Mémoire[$2000] = 0F

8. Registres du Motorola 6809

Registre	Taille	Description	Valeur Initiale
PC	16 bits	Program Counter - Prochaine instruction	1400
A	8 bits	Accumulator A - Opérations arithmétiques	00
B	8 bits	Accumulator B - Complémentaire à A	00
D	16 bits	Double Accumulator (A:B)	0000
X	16 bits	Index Register X - Adressage indexé	0000
Y	16 bits	Index Register Y - Second registre d'index	0000
U	16 bits	User Stack Pointer - Pile utilisateur	0000
S	16 bits	System Stack Pointer - Pile système	0000
DP	8 bits	Direct Page - Page pour adressage direct	00

Registre de Condition (CCR) - 8 Flags:

Flag	Bit	Nom	Description
E	7	Entire	État complet sauvegardé lors d'interruption
F	6	FIRQ Mask	Masque d'interruptions FIRQ
H	5	Half Carry	Retenue du bit 3 au bit 4 (BCD)
I	4	IRQ Mask	Masque d'interruptions IRQ
N	3	Negative	Résultat négatif (bit 7 = 1)
Z	2	Zero	Résultat égal à zéro
V	1	Overflow	Dépassement arithmétique signé
C	0	Carry	Retenue ou emprunt

9. Jeu d'Instructions Principal

9.1 Instructions de Transfert

Mnémonique	Description	Exemple
LDA	Load Accumulator A	LDA #$05
LDB	Load Accumulator B	LDB $2000
LDD	Load Double (A:B)	LDD #$1234
LDX	Load Index X	LDX #$3000
LDY	Load Index Y	LDY ,X
STA	Store Accumulator A	STA $2000
STB	Store Accumulator B	STB $2001
STD	Store Double	STD $3000
STX	Store Index X	STX $4000
STY	Store Index Y	STY $4002

9.2 Instructions Arithmétiques

Mnémonique	Description	Exemple	Flags
ADDA	Add to A	ADDA #$05	N,Z,V,C,H
ADDB	Add to B	ADDB $20	N,Z,V,C,H
ADDD	Add to D (16-bit)	ADDD #$1000	N,Z,V,C
SUBA	Subtract from A	SUBA #$03	N,Z,V,C
SUBB	Subtract from B	SUBB $30	N,Z,V,C
SUBD	Subtract from D	SUBD #$0500	N,Z,V,C
MUL	Multiply A × B → D	MUL	Z,C
DAA	Decimal Adjust A	DAA	N,Z,C
INCA	Increment A	INCA	N,Z,V
INCB	Increment B	INCB	N,Z,V
DECA	Decrement A	DECA	N,Z,V
DECB	Decrement B	DECB	N,Z,V

9.3 Instructions Logiques

Mnémonique	Description	Exemple	Flags
ANDA	AND with A	ANDA #$0F	N,Z,V=0
ANDB	AND with B	ANDB $40	N,Z,V=0
ORA	OR with A	ORA #$F0	N,Z,V=0
ORB	OR with B	ORB $50	N,Z,V=0
EORA	XOR with A	EORA #$FF	N,Z,V=0
EORB	XOR with B	EORB $60	N,Z,V=0
COMA	Complement A	COMA	N,Z,V=0,C=1
COMB	Complement B	COMB	N,Z,V=0,C=1
NEGA	Negate A	NEGA	N,Z,V,C
NEGB	Negate B	NEGB	N,Z,V,C

9.4 Instructions de Branchement

Mnémonique	Condition	Description
BRA	Toujours	Branch Always (saut inconditionnel)
BEQ	Z = 1	Branch if Equal (si égal)
BNE	Z = 0	Branch if Not Equal (si différent)
BMI	N = 1	Branch if Minus (si négatif)
BPL	N = 0	Branch if Plus (si positif)
BCS	C = 1	Branch if Carry Set
BCC	C = 0	Branch if Carry Clear
BVS	V = 1	Branch if Overflow Set
BVC	V = 0	Branch if Overflow Clear
BGT	Z=0 ET (N⊕V)=0	Branch if Greater Than (signé)
BLT	(N⊕V)=1	Branch if Less Than (signé)
BGE	(N⊕V)=0	Branch if Greater or Equal (signé)
BLE	Z=1 OU (N⊕V)=1	Branch if Less or Equal (signé)

9.5 Instructions de Pile

PSHS / PSHU - Push sur pile S ou U:
PSHS A,B,X,Y ; Empile A, B, X, Y sur la pile système
PSHU D,X ; Empile D et X sur la pile utilisateur

PULS / PULU - Pull de la pile S ou U:
PULS A,B,X,Y ; Dépile vers A, B, X, Y
PULU D,X ; Dépile vers D et X

Ordre d'empilement (PSHS ALL):
PC, U, Y, X, DP, B, A, CC

Ordre de dépilement (PULS ALL):
CC, A, B, DP, X, Y, U, PC

9.6 Instructions de Contrôle

Mnémonique	Description	Action
NOP	No Operation	Aucune opération (1 cycle)
SWI	Software Interrupt	Interruption logicielle
RTI	Return from Interrupt	Retour d'interruption
RTS	Return from Subroutine	Retour de sous-routine
JSR	Jump to Subroutine	Saut vers sous-routine
JMP	Jump	Saut inconditionnel
BSR	Branch to Subroutine	Branchement vers sous-routine

10. Organisation de la Mémoire

Carte Mémoire du Motorola 6809 (64 KB total):

0x0000 - 0x00FF : Page Directe (256 octets)
• Accès rapide via adressage direct
• Contrôlée par le registre DP

0x0100 - 0x1DFF : RAM Utilisateur (7424 octets)
• Zone libre pour programmes et données

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

0x1F00 - 0x1FFF : Pile Système S (256 octets)
• Réservée pour la pile S
• PSHS/PULS, JSR/RTS, interruptions

0x2000 - 0x7FFF : RAM Étendue (~24 KB)
• Données et programmes volumineux
• Zone la plus grande

0x8000 - 0xBFFF : Zone ROM/Programme (~16 KB)
• Code programme principal
• Lecture seule en production

0xC000 - 0xFEFF : Périphériques I/O (~12 KB)
• Mappés en mémoire (Memory-Mapped)
• Ports, timers, UART, etc.

0xFF00 - 0xFFFF : Vecteurs d'Interruption (256 octets)
• 0xFFF0-0xFFF1: Reserved
• 0xFFF2-0xFFF3: SWI3
• 0xFFF4-0xFFF5: SWI2
• 0xFFF6-0xFFF7: FIRQ
• 0xFFF8-0xFFF9: IRQ
• 0xFFFA-0xFFFB: SWI
• 0xFFFC-0xFFFD: NMI
• 0xFFFE-0xFFFF: RESET

11. Techniques de Débogage

11.1 Exécution Pas à Pas

L'exécution pas à pas est l'outil principal de débogage. Elle permet de:
• Contrôler l'exécution instruction par instruction
• Observer les changements de registres après chaque instruction
• Suivre le flux d'exécution du programme
• Identifier où se produisent les erreurs

Procédure:
1. Assembler le programme
2. Ouvrir la fenêtre Architecture pour voir les registres
3. Cliquer sur "→ Pas à pas" pour exécuter une instruction
4. Observer les changements (registres surlignés en vert)
5. Répéter jusqu'à la fin du programme

11.2 Points d'Arrêt

Les points d'arrêt permettent de suspendre l'exécution à des adresses spécifiques:
• Utiles pour les programmes longs
• Permettent d'examiner l'état à des moments précis
• Évitent d'exécuter pas à pas tout le programme

Configuration:
1. Identifier l'adresse où placer le point d'arrêt
2. Le définir via l'interface du débogueur
3. Lancer l'exécution normale (■ Démarrer)
4. Le programme s'arrête automatiquement au point d'arrêt

11.3 Inspection de la Mémoire

La visualisation de la mémoire aide à:
• Vérifier que les données sont stockées correctement
• Détecter les corruptions de mémoire
• Suivre l'évolution des données pendant l'exécution

Méthode:
1. Ouvrir la fenêtre RAM ou ROM
2. Localiser l'adresse à examiner
3. Observer les valeurs en hexadécimal
4. Utiliser la colonne ASCII pour identifier du texte
5. Comparer avant/après l'exécution d'instructions

11.4 Erreurs Courantes et Solutions

Erreur	Symptôme	Solution
Programme ne démarre pas	PC reste à 0000	Vérifier directive ORG et assemblage
Résultat incorrect	Valeurs inattendues	Exécution pas à pas pour identifier l'étape
Mémoire corrompue	Valeurs bizarres	Vérifier adresses STA/STB/STD
Boucle infinie	Programme ne termine pas	Vérifier conditions de branchement
Débordement de pile	Comportement erratique	Équilibrer PUSH et PULL

12. Conclusion

Le Simulateur Motorola 6809 représente un outil pédagogique et professionnel complet pour l'apprentissage et le développement sur cette architecture de microprocesseur historique. Ce projet démontre l'importance de comprendre les fondements de l'architecture des ordinateurs, concepts qui restent pertinents dans les systèmes modernes malgré les évolutions technologiques.

12.1 Objectifs Atteints

✓ **Émulation complète** du jeu d'instructions du Motorola 6809
✓ **Débogueur intégré** avec points d'arrêt et exécution pas à pas
✓ **Interface graphique intuitive** facilitant l'utilisation
✓ **Visualisation en temps réel** des registres et de la mémoire
✓ **Documentation complète** avec algorithmes détaillés
✓ **Exemples pratiques** pour faciliter l'apprentissage

12.2 Apports Pédagogiques

Ce simulateur permet aux étudiants de:
• Comprendre concrètement le fonctionnement d'un microprocesseur
• Visualiser l'exécution du code assembleur instruction par instruction
• Expérimenter avec différents modes d'adressage
• Développer des compétences en débogage de bas niveau
• Appréhender les concepts de registres, flags et gestion mémoire

12.3 Perspectives d'Évolution

Améliorations futures possibles:
• Extension du jeu d'instructions avec opcodes supplémentaires
• Simulation de périphériques I/O (affichage, clavier)
• Débogueur avancé avec watchpoints sur mémoire
• Profileur de performance pour analyser les temps d'exécution
• Export/import de programmes au format standard
• Mode multi-fichiers pour projets complexes
• Intégration avec émulateurs de systèmes complets

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.

Annexes

A. Table des Opcodes (Sélection)

Opcode	Mnémonique	Mode	Cycles
86	LDA	Immédiat	2
96	LDA	Direct	4
B6	LDA	Étendu	5
C6	LDB	Immédiat	2
D6	LDB	Direct	4
F6	LDB	Étendu	5
CC	LDD	Immédiat	3
DC	LDD	Direct	5
FC	LDD	Étendu	6
8B	ADDA	Immédiat	2
9B	ADDA	Direct	4
BB	ADDA	Étendu	5
3D	MUL	Inhérent	11
12	NOP	Inhérent	2
3F	SWI	Inhérent	19
20	BRA	Relatif	3
27	BEQ	Relatif	3
26	BNE	Relatif	3

B. Raccourcis Clavier (Proposés)

Édition:
Ctrl+N : Nouveau fichier
Ctrl+O : Ouvrir fichier
Ctrl+S : Sauvegarder
Ctrl+A : Sélectionner tout

Exécution:
F5 : Assembler
F9 : Démarrer/Continuer
F10 : Pas à pas
F11 : Pas à pas détaillé
Shift+F5 : Arrêter
Ctrl+F5 : Réinitialiser

Débogage:
F8 : Toggle point d'arrêt
Ctrl+F8 : Liste des points d'arrêt

Université Hassan Premier de Settat
Faculté des Sciences et Techniques de Settat
Année Universitaire 2025-2026

Rapport généré le 25/12/2025 à 06:14