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. La Console Principale Interface centrale de contrôle
2. Les fenêtres supplémentaires peuvent être ouvertes via les boutons dédiés :
    1. Éditeur Avancé
    2. Architecture
    3. RAM
    4. ROM

3. Vue d'ensemble de l'interface

3.1 Console Principale

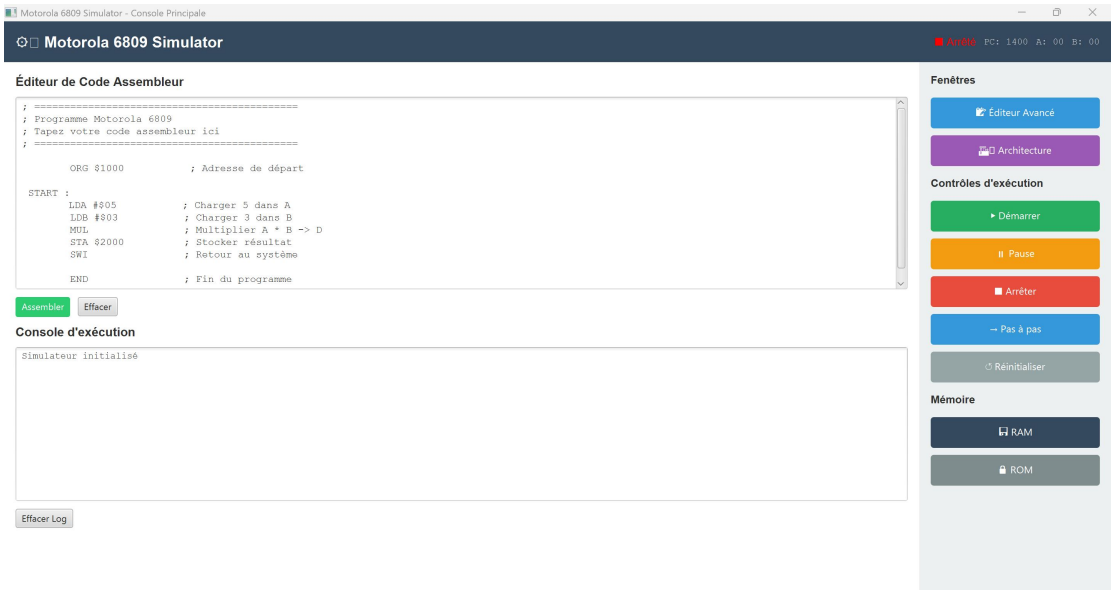


Figure 1 : Console principale du simulateur

La console principale est l'interface centrale qui intègre tous les composants du simulateur.

A. Barre Supérieure

Composants affichés :

- **Logo** : Motorola 6809 Simulator
- **État d'exécution** :
  - ■ Arrêté (texte rouge)
  - ▶ En cours (texte vert)
  - ⏸ Pause (texte orange)
- **Registres rapides** : PC, A, B (valeurs hexadécimales)

Exemple d'affichage :

Motorola 6809 Simulator ■ Arrêté PC: 1400 A: 00 B: 00

B. Section Éditeur de Code Assembleur (Panneau Gauche)

Zone de saisie principale :

- Police monospace pour meilleure lisibilité
- Support complet des commentaires (lignes commençant par ;)
- Dimensions : zone large pour écriture confortable
- Code exemple fourni par défaut

Boutons disponibles :

1. **Assembler (Vert)**

    - Compile le code assembleur en code machine
    - Vérifie la syntaxe
    - Affiche les résultats dans la console
    - Charge le programme en mémoire ROM

2. **Effacer**

    - Vide complètement la zone de texte
    - Prépare pour un nouveau programme

Code exemple par défaut :

assembly
; ==========; Programme Motorola 6809; Tapez votre code assembleur ici; ========== ORG $1000 ; Adresse de départSTART LDA #$05 ; Charger 5 dans A LDB #$03 ; Charger 3 dans B MUL ; Multiplier A * B -> D STA $2000 ; Stocker résultat SWI ; Retour au système END ; Fin du programme

C. Console d'Exécution (Panneau Inférieur)

Fonction :

- Affiche les messages d'assemblage (succès/erreur)
- Affiche les logs d'exécution
- Affiche les messages système
- Affiche les résultats des opérations

État initial :

Simulateur initialisé

Messages typiques :

✓ Programme assemblé avec succèsTaille: 8 octetsAdresse de départ: $1400✓ Exécution terminéePC final: $1408

Bouton disponible :

- **Effacer Log** : Nettoie complètement la console

D. Panneau Fenêtres (Panneau Droit - Section Haute)

Cette section permet d'ouvrir les fenêtres auxiliaires.

Boutons disponibles :

1. **Éditeur Avancé (Bleu)**
    - Ouvre l'éditeur de code dédié dans une fenêtre séparée
    - Offre plus d'espace pour l'édition
    - Fonctionnalités avancées d'exécution
2. **Architecture (Violet)**
    - Ouvre la vue détaillée des registres
    - Affichage en temps réel de tous les registres
    - Visualisation du registre de condition (CCR)

E. Contrôles d'Exécution (Panneau Droit - Section Centre)

Les boutons de contrôle principaux pour gérer l'exécution du programme.

▶ **Démarrer (Vert)**

- Lance l'exécution continue du programme
- Le programme s'exécute jusqu'à rencontrer :
    - Une instruction SWI
    - Un point d'arrêt
    - La fin du programme
- Change l'état en "▶ En cours"

⏸ **Pause (Orange)**

- Suspend temporairement l'exécution en cours
- Fige l'état du système pour inspection
- Permet d'examiner les registres et la mémoire
- Cliquer à nouveau pour reprendre

■ **Arrêter (Rouge)**

- Arrête complètement l'exécution
- Réinitialise l'état à "Arrêté"
- Le programme peut être relancé depuis le début

→ **Pas à pas (Bleu)**

- Exécute une seule instruction à la fois
- Idéal pour le débogage ligne par ligne
- Met à jour l'affichage après chaque instruction
- Permet de suivre précisément le flux d'exécution

↺ **Réinitialiser (Gris)**

- Remet tous les registres à zéro
- Efface la mémoire
- Réinitialise le compteur de programme
- Recharge le programme si assemblé

F. Section Mémoire (Panneau Droit - Section Basse)

Accès rapide aux vues mémoire.

Boutons disponibles :

1. **RAM (Bleu foncé)**
    - Ouvre la fenêtre de visualisation de la RAM
    - Affiche les adresses 0000 et suivantes
    - Permet de voir les données en temps réel
2. **ROM (Gris)**
    - Ouvre la fenêtre de visualisation de la ROM
    - Affiche le code programme compilé
    - Zone en lecture seule

4. Éditeur de code

4.1 Fenêtre Éditeur Assembleur 6809

Figure 2 : Fenêtre Éditeur de code

L'éditeur dédié offre un espace optimisé pour l'écriture et l'exécution de code assembleur.

Titre

Éditeur de Code Assembleur Motorola 6809

Composants de l'Éditeur

Barre d'outils (5 boutons) :

1. Nouveau (Bleu)

- Efface tout le contenu de l'éditeur
- Réinitialise l'environnement
- Prépare pour un nouveau programme
- Raccourci rapide pour recommencer

2. Assembler (Vert)

- Compile le code assembleur en code machine
- Vérifie la syntaxe de chaque ligne
- Génère les opcodes correspondants
- Charge le programme en mémoire ROM
- Affiche un message de confirmation ou d'erreur

3. Pas à pas (Orange)

- Compile automatiquement le code si nécessaire
- Exécute une seule instruction
- Met à jour tous les registres
- Affiche l'état dans la fenêtre Architecture
- Permet un débogage précis

4. Exécuter (Rouge)

- Compile automatiquement le code si nécessaire
- Lance l'exécution complète du programme
- Continue jusqu'à :
    - Instruction SWI
    - Fin du programme
    - Erreur d'exécution
- Le bouton bascule en "Arrêter" pendant l'exécution

5. Reset (Violet)

- Réinitialise complètement le simulateur
- Remet tous les registres à zéro
- Efface la mémoire
- Charge le vecteur RESET (0xFFFE)
- Prépare pour une nouvelle exécution

Zone de Texte

Caractéristiques :

- Grande zone d'édition (dimensions optimisées)
- Police monospace (Courier/Monospaced) pour alignement
- Support complet des commentaires
- Défilement automatique vertical et horizontal
- Numérotation implicite des lignes

Barre d'État

Affiche l'état actuel de l'éditeur :

- **"Prêt"** : En attente d'action
- **"Assemblage en cours..."** : Pendant la compilation
- **"Exécution..."** : Programme en cours
- **"Terminé"** : Exécution terminée

4.2 Syntaxe du Code Assembleur

Structure d'un Programme

assembly
; 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)

Règles de formatage :

- Les commentaires commencent par ;
- Les étiquettes se terminent par :
- Les instructions sont indentées (espaces ou tabulations)
- Les directives (ORG, END) peuvent être à gauche ou indentées

Directives Assembleur

ORG (Origin)

Définit l'adresse de départ du programme en mémoire.

assembly
ORG $1000 ; Démarrer à l'adresse $1000ORG $1400 ; Démarrer à l'adresse $1400ORG $C000 ; Démarrer à l'adresse $C000

Utilisation :

- Obligatoire au début de chaque programme
- Peut être n'importe quelle adresse valide (0000-FFFF)
- Détermine où le code sera chargé en mémoire

END

Marque la fin du programme.

assembly
END ; Fin du programme

Utilisation :

- Obligatoire à la fin du programme
- Indique à l'assembleur d'arrêter la compilation
- Aucun code ne doit suivre cette directive

Modes d'Adressage

Le Motorola 6809 supporte plusieurs modes d'adressage.

1. Immédiat (#)

La valeur est directement dans l'instruction.

assembly
LDA #$05 ; Charge la valeur 05 dans ALDB #$FF ; Charge la valeur FF dans BLDD #$1234 ; Charge la valeur 1234 dans D (16 bits)LDX #$2000 ; Charge la valeur 2000 dans X

Caractéristiques :

- Symbole : #
- Le plus rapide (pas d'accès mémoire)
- La donnée est dans l'instruction elle-même

2. Direct ($)

Adresse dans la page directe (0000-00FF).

assembly
LDA $50 ; Charge depuis l'adresse de page directe $0050STA $80 ; Stocke à l'adresse de page directe $0080

Caractéristiques :

- Pour adresses 00-FF seulement
- Plus rapide que l'adressage étendu
- Économise un octet

3. Étendu (Extended)

Adresse complète 16 bits.

assembly
LDA $2000 ; Charge depuis l'adresse $2000STA $3000 ; Stocke à l'adresse $3000LDX $4000 ; Charge X depuis $4000

Caractéristiques :

- Accès à toute la mémoire (0000-FFFF)
- Plus lent que l'adressage direct
- Utilise 3 octets (opcode + adresse 16-bit)

4. Indexé

Utilise un registre d'index pour calculer l'adresse.

assembly
LDA ,X ; Charge depuis l'adresse pointée par XSTA ,Y ; Stocke à l'adresse pointée par YLDB 5,X ; Charge depuis X+5

Caractéristiques :

- Très flexible
- Idéal pour les tableaux
- Plusieurs variantes possibles

5. Inhérent

Pas d'opérande, l'instruction contient tout.

assembly
NOP ; Aucune opérationMUL ; Multiplier A × B → DINCA ; Incrémenter ADECB ; Décrémenter B

Caractéristiques :

- Le plus compact (1 octet)
- Opération sur registres implicites
- Le plus rapide

5. Architecture et registre

5.1 Fenêtre "Architecture 6809 - Vue Dynamique"

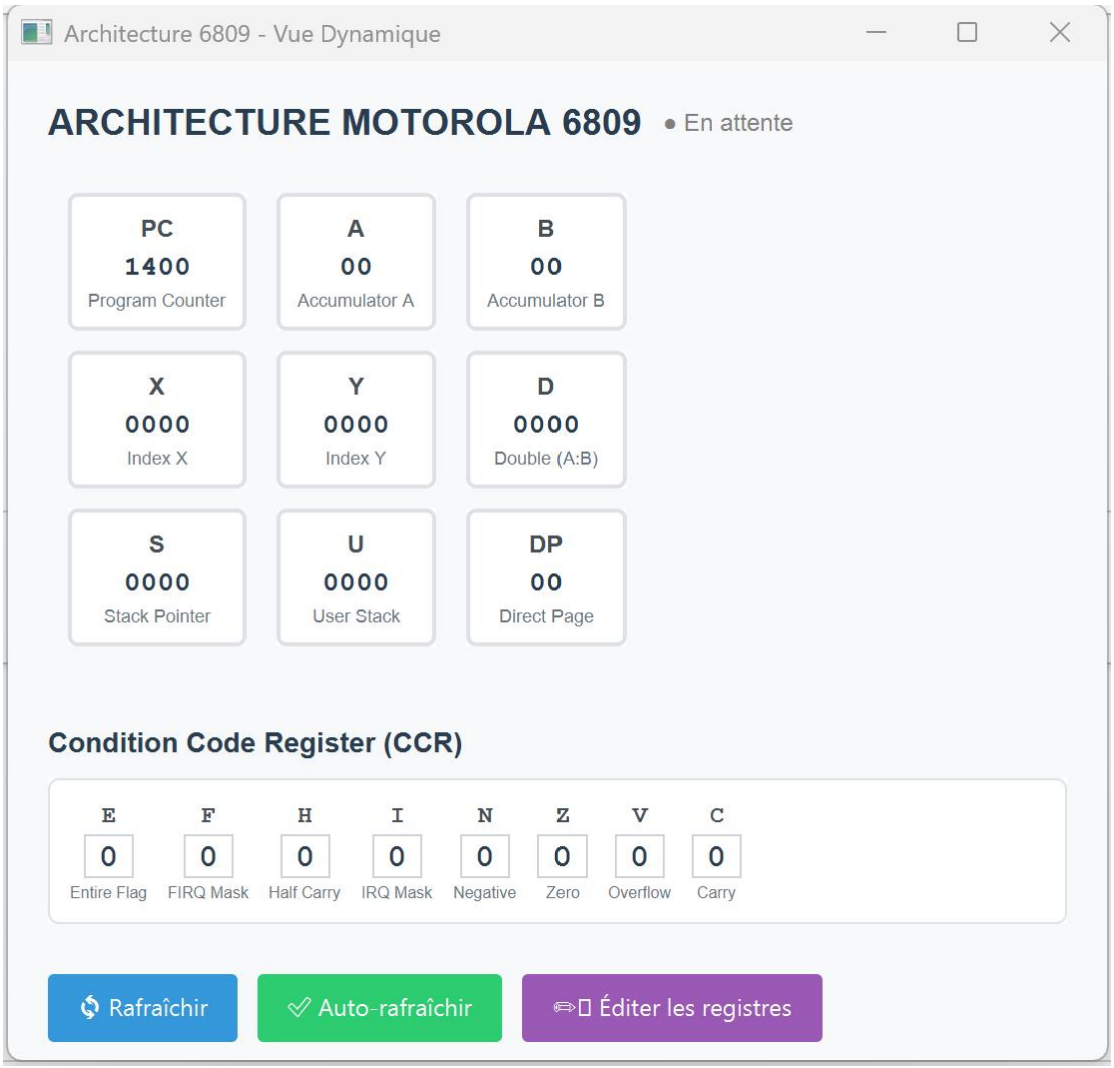


Figure 3 : Fenêtre Architecture 6809 - Vue Dynamique

Cette fenêtre offre une visualisation complète et dynamique de tous les registres du processeur en temps réel.

Titre et État

Format du titre :

ARCHITECTURE MOTOROLA 6809 ● En attente

États possibles :

- ● **En attente** (gris) : Simulateur arrêté, en attente de commande
- ● **En cours** (vert) : Programme en cours d'exécution
- ● **En pause** (orange) : Exécution suspendue temporairement

Format d'affichage de chaque registre :

- Nom du registre (grande police, gras)
- Valeur en hexadécimal (très grande police)
- Description textuelle (petite police)

Mise à Jour Visuelle

Animations en temps réel :

1. **Registres modifiés :**
    - Surlignés en **vert clair** (#e8f5e8)
    - Bordure **verte** (#4CAF50)
    - Durée : 300 millisecondes
    - Puis retour à l'affichage normal
2. **Flags modifiés :**
    - Bordure **orange** (#FF9800)
    - Durée : 200 millisecondes
    - Texte vert si flag = 1
    - Texte noir si flag = 0
3. **Titre dynamique :**
    - Change selon l'état d'exécution
    - Couleur adaptée à l'état
    - Mise à jour en temps réel

5.2 Registres Principaux

PC (Program Counter) - 16 bits

Valeur affichée :

- **Initiale :** 0000
- **Après assemblage :** Adresse ORG (ex: 1400)
- **Format :** 4 chiffres hexadécimaux

Description : Program Counter

Fonction :

- Pointe vers la **prochaine** instruction à exécuter
- Incrémenté automatiquement après chaque instruction
- Modifié par les instructions de saut (JMP, JSR, BRA, etc.)

Exemple de progression :

Initial: PC = 1400Après LDA #$05: PC = 1402 (2 octets: opcode + valeur)Après MUL: PC = 1403 (1 octet)Après STA $2000: PC = 1406 (3 octets: opcode + adresse)

A (Accumulator A) - 8 bits

Valeur affichée :

- **Initiale :** 00
- **Plage :** 00 à FF (0 à 255 décimal)
- **Format :** 2 chiffres hexadécimaux

Description : Accumulator A

Fonction :

- Registre principal pour opérations arithmétiques
- Registre principal pour opérations logiques
- Partie **haute** du registre D (16 bits)
- Utilisé pour multiplication (A × B → D)

Opérations typiques :

assembly
LDA #$05 ; Charger une valeurADDA #$10 ; AdditionINCA ; IncrémentationSTA $2000 ; Stocker en mémoire

B (Accumulator B) - 8 bits

Valeur affichée :

- **Initiale :** 00
- **Plage :** 00 à FF
- **Format :** 2 chiffres hexadécimaux

Description : Accumulator B

Fonction :

- Second accumulateur, complémentaire à A
- Partie **basse** du registre D (16 bits)
- Utilisé pour multiplication (A × B → D)

Relation avec D :

Si A = 12 et B = 34Alors D = 1234

X (Index Register X) - 16 bits

Valeur affichée :

- **Initiale :** 0000
- **Plage :** 0000 à FFFF
- **Format :** 4 chiffres hexadécimaux

Description : Index X

Fonction :

- Registre d'index principal
- Utilisé pour l'adressage indexé
- Manipulation de tableaux et structures
- Peut être incrémenté/décrémenté

Exemple d'utilisation :

assembly
LDX #$2000 ; X pointe vers $2000LDA ,X ; Charger depuis l'adresse dans
XLDA 5,X ; Charger depuis X+5

Y (Index Register Y) - 16 bits

Valeur affichée :

- **Initiale :** 0000
- **Plage :** 0000 à FFFF
- **Format :** 4 chiffres hexadécimaux

Description : Index Y

Fonction :

- Second registre d'index
- Même fonctionnalité que X
- Permet des opérations sur deux tableaux simultanément

D (Double Accumulator) - 16 bits

Valeur affichée :

- **Initiale :** 0000

- **Plage :** 0000 à FFFF
- **Format :** 4 chiffres hexadécimaux

Description : Double (A:B)

Fonction :

- Registre composite formé de A (haute) et B (basse)
- Utilisé pour opérations 16 bits
- Résultat de la multiplication (MUL)

Calcul automatique :

D = (A << 8) | BD = A × 256 + BExemple:A = 12 (hex) = 18 (déc)B = 34 (hex) = 52 (déc)D = 1234 (hex) = 4660 (déc)

S (Stack Pointer) - 16 bits

Valeur affichée :

- **Initiale :** 0000 (ou valeur configurée)
- **Typique :** 1FFF (haut de la pile système)
- **Format :** 4 chiffres hexadécimaux

Description : Stack Pointer

Fonction :

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

Comportement :

PUSH: S = S - 1, puis écriture à [S]PULL: lecture depuis [S], puis S = S + 1

U (User Stack Pointer) - 16 bits

Valeur affichée :

- **Initiale :** 0000 (ou valeur configurée)
- **Typique :** 1E00 (pile utilisateur)
- **Format :** 4 chiffres hexadécimaux

Description : User Stack

Fonction :

- Pointeur de pile utilisateur
- Séparé de la pile système (S)
- Permet gestion de contextes multiples
- Utilisé par PSHU/PULU

DP (Direct Page) - 8 bits

Valeur affichée :

- **Initiale :** 00
- **Plage :** 00 à FF
- **Format :** 2 chiffres hexadécimaux

Description : Direct Page

Fonction :

- Définit la page pour l'adressage direct
- Permet accès rapide à 256 octets
- Économise un octet par instruction
- Forme l'adresse complète : DP × 256 + offset

Exemple :

DP = 20 (hex)LDA $50 (direct)→ Adresse réelle = $2050

5.3 Registre de Condition (CCR)

Titre de la section :

Condition Code Register (CCR)

Le registre CC contient 8 flags sur 8 bits qui reflètent l'état du processeur.

Affichage des Flags

Disposition horizontale :

Format de chaque flag :

- Nom (1 lettre, en haut)
- Valeur (0 ou 1, grande police)
- Description (en bas, petite police)

Mise à jour visuelle :

- **Valeur 0 :** Texte noir, fond blanc
- **Valeur 1 :** Texte vert (#2ecc71), fond blanc
- **Lors du changement :** Bordure orange 200ms

Description Détaillée des Flags

E - Entire Flag (Bit 7, masque 0x80)

Description : Entire Flag

Signification :

- **1 :** État complet sauvegardé lors d'une interruption
- **0 :** État partiel sauvegardé (FIRQ uniquement)

Utilisation :

- Indique quel type de sauvegarde a été fait
- IRQ/NMI/SWI : E=1 (sauvegarde complète)
- FIRQ : E=0 (sauvegarde partielle: PC, CC)

F - FIRQ Mask (Bit 6, masque 0x40)

Description : FIRQ Mask

Signification :

- **1 :** Interruptions FIRQ désactivées (masquées)
- **0 :** Interruptions FIRQ activées

Utilisation :

- Contrôle les Fast Interrupts (FIRQ)
- Modifié par instructions ORCC/ANDCC
- Automatiquement mis à 1 lors d'un FIRQ

H - Half Carry (Bit 5, masque 0x20)

Description : Half Carry

Signification :

- **1 :** Retenue du bit 3 au bit 4
- **0 :** Pas de demi-retenue

Utilisation :

- Spécifique à l'arithmétique BCD
- Utilisé par l'instruction DAA
- Indique retenue sur 4 bits (demi-octet)

Exemple :

0F (15)+ 01 (1)---- 10 (16) → H=1 (retenue bit3→bit4)

I - IRQ Mask (Bit 4, masque 0x10)

Description : IRQ Mask

Signification :

- **1 :** Interruptions IRQ désactivées (masquées)
- **0 :** Interruptions IRQ activées

Utilisation :

- Contrôle les interruptions normales (IRQ)
- Protection des sections critiques
- Modifié par ORCC/ANDCC/CLI/SEI

N - Negative (Bit 3, masque 0x08)

Description : Negative

Signification :

- **1 :** Résultat négatif (bit 7 = 1)
- **0 :** Résultat positif (bit 7 = 0)

Utilisation :

- Indique le signe en complément à 2
- Testé par les branchements BMI/BPL
- Mis à jour par la plupart des opérations

Exemple :

LDA #$FF ; FF = -1 en signé → N=1LDA #$7F ; 7F = +127 → N=0

Z - Zero (Bit 2, masque 0x04)

Description : Zero

Signification :

- **1 :** Résultat égal à zéro
- **0 :** Résultat différent de zéro

Utilisation :

- Le flag le plus utilisé !
- Testé par BEQ (Branch if Equal) et BNE (Branch if Not Equal)
- Mis à jour par toutes les opérations arithmétiques/logiques

Exemple :

LDA #$00 ; Z=1 (résultat est zéro)LDA #$05 ; Z=0 (résultat non-zéro)SUBA #$05 ; A=0 → Z=1

V - Overflow (Bit 1, masque 0x02)

Description : Overflow

Signification :

- **1 :** Dépassement arithmétique signé détecté
- **0 :** Pas de dépassement

Utilisation :

- Important pour arithmétique signée
- Détecte les erreurs de calcul en complément à 2
- Testé par BVS/BVC

Exemple :

LDA #$7F ; 127 (max positif signé)ADDA #$01 ; +1 ; Résultat: 80 hex = 128 non-signé ; Mais -128 en signé → V=1 (overflow!)

C - Carry (Bit 0, masque 0x01)

Description : Carry

Signification :

- **1 :** Retenue générée ou emprunt
- **0 :** Pas de retenue

Utilisation :

- Arithmétique non-signée
- Opérations multi-octets
- Décalages et rotations
- Testé par BCS/BCC

Exemple :

```
LDA #$FF    ; 255ADDA #$01   ; +1            ; Résultat: 00 (256 mod 256)          ;
C=1 (retenue générée)
```

5.4 Boutons de Contrôle de la Fenêtre

Rafraîchir (Bleu)

Fonction :

- Met à jour manuellement tous les registres
- Lit les valeurs actuelles du backend
- Force une mise à jour complète de l'affichage

Utilisation :

- Utile si l'auto-rafraîchissement est désactivé
- Vérifier l'état à un moment précis
- Résoudre problèmes d'affichage

✓ Auto-rafraîchir (Vert/Rouge)

État ACTIVÉ (bouton vert ✓) :

- Mise à jour automatique en temps réel
- Rafraîchissement après chaque instruction
- Animation des changements
- Mode recommandé pour observation

État DÉSACTIVÉ (bouton rouge ○) :

- Pas de mise à jour automatique
- Économise ressources processeur

- Utiliser "Rafraîchir" pour voir les changements

Basculement :

- Cliquer pour activer/désactiver
- Change couleur et icône
- Persistant pendant la session

Éditer les registres (Violet)

Fonction :

- Ouvre une fenêtre d'édition modale
- Permet modification manuelle des registres
- Utile pour tests et débogage avancé

Fenêtre d'édition :

Champs disponibles :

PC: [1400]A: [00]B: [00]X: [0000]Y: [0000]S: [0000]U: [0000]DP: [00]

Format :

- Valeurs en hexadécimal
- Accepte avec ou sans préfixe $ ou 0x
- Validation automatique

Boutons :

- **Appliquer** : Sauvegarde les modifications
- **Annuler** : Ferme sans modifier

Utilisation typique :

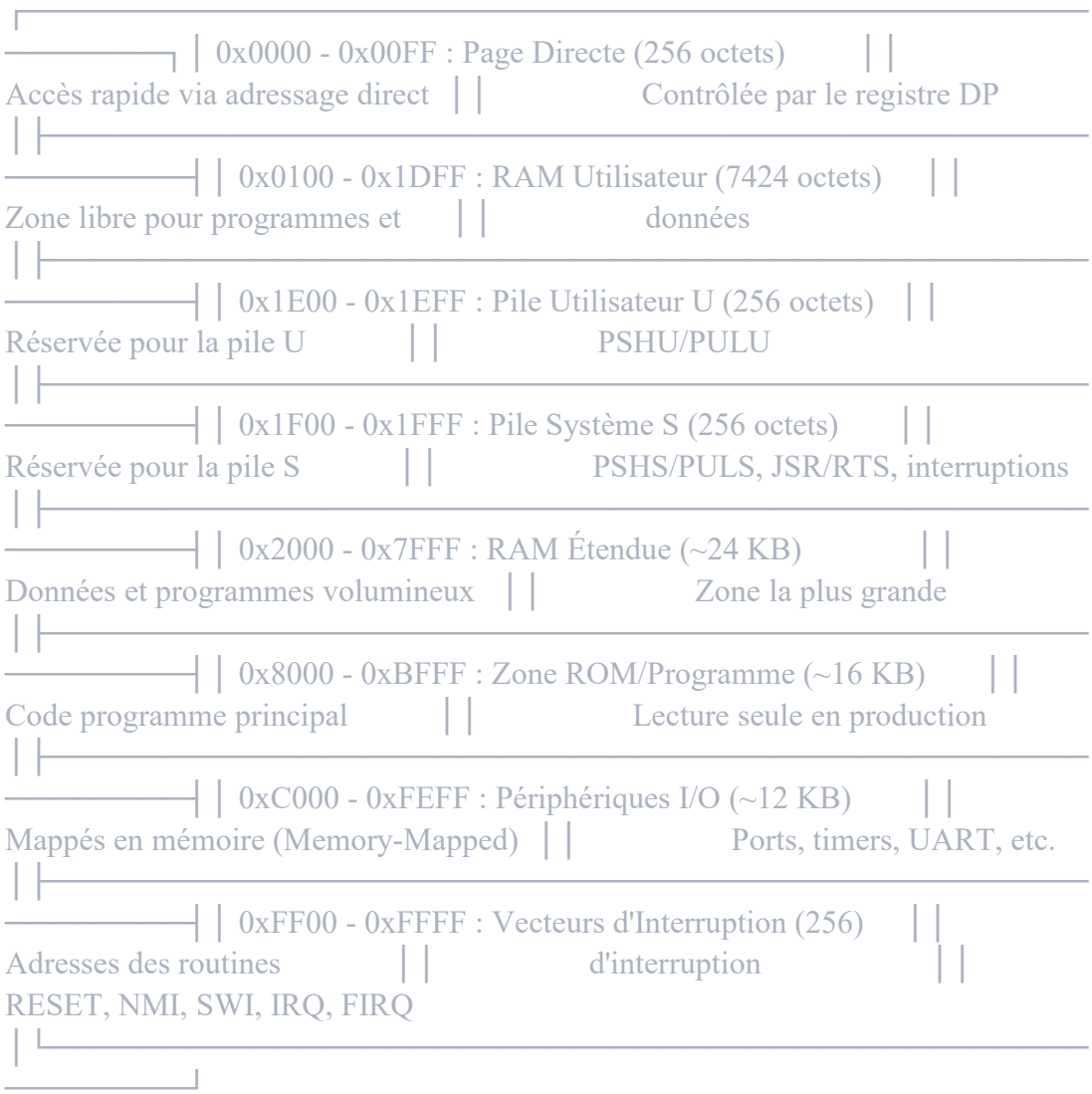
1. Cliquer sur "Éditer les registres"
2. Modifier les valeurs désirées
3. Cliquer sur "Appliquer"
4. Les registres sont mis à jour immédiatement

6. Gestion de la mémoire

6.1 Organisation de la Mémoire

Le Motorola 6809 dispose d'un espace d'adressage de **64 KB** (65536 octets) allant de **0x0000** à **0xFFFF**.

Carte Mémoire Standard



Vecteurs d'Interruption

6.2 Fenêtre Vue Mémoire - RAM

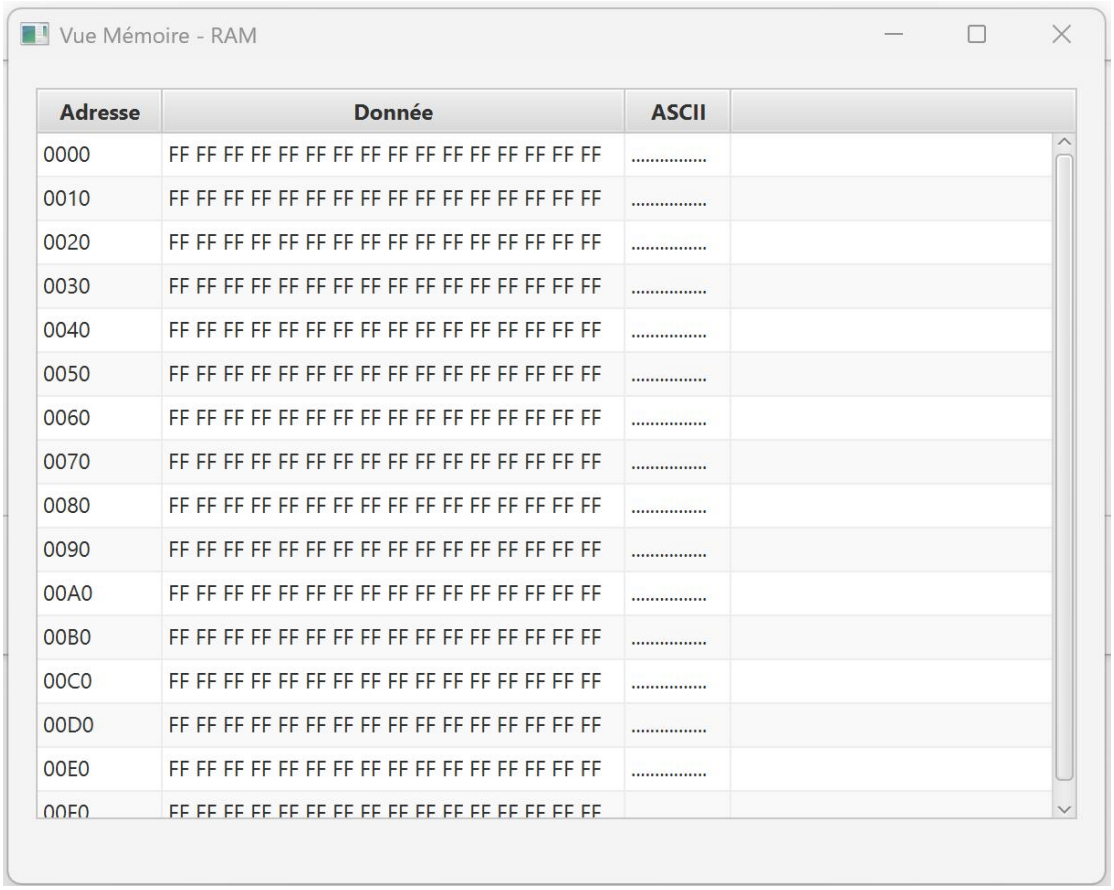


Figure 4 : Fenêtre Vue Mémoire - RAM

La fenêtre RAM affiche le contenu de la mémoire vive (Read-Write Memory).

Titre

Vue Mémoire - RAM

Caractéristiques d'Affichage

Format en tableau à 3 colonnes :

Adresse	**Donnée**	**ASCII**
0000	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	.....
0010	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	.....
0020	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	.....
...	...	...

Colonne Adresse :

- Format : 4 chiffres hexadécimaux
- Alignée à gauche

- Commence à 0000
- Incrémente par 16 (10 hex) à chaque ligne

Colonne Donnée :

- 16 octets par ligne en hexadécimal
- Format : FF FF FF ... (séparés par espaces)
- Police monospace pour alignement
- Largeur maximale de la colonne

Colonne ASCII :

- Représentation ASCII des 16 octets
- Caractères imprimables (32-126) : affichés tels quels
- Caractères non-imprimables : point (.)
- Police monospace
- Largeur fixe : 16 caractères

Initialisation et Contenu

État initial :

- Toutes les valeurs à **FF** (255 décimal)
- Ceci indique mémoire non initialisée
- Changent après exécution d'instructions de stockage

Après exécution :

Adresse | Donnée--------+----------2000 | 05 03 0F 00 00 00 ... | Valeurs stockées par le programme

Taille et Navigation

Plage affichée :

- Commence à 0000
- Typiquement 256 lignes × 16 octets = 4096 octets visibles
- Couvre les adresses 0000 à 0FFF

Navigation :

- Barre de défilement verticale
- Permet de voir toutes les adresses jusqu'à FFFF
- Utiliser molette souris ou scrollbar

Redimensionnement :

- Fenêtre redimensionnable
- Colonnes ajustables
- Maximiser pour voir plus de lignes

Utilisation Pratique

Visualisation :

1. Cliquer sur **RAM** dans la console principale
2. La fenêtre s'ouvre avec le contenu actuel
3. Défiler pour trouver l'adresse désirée

Lecture :

Exemple:Adresse 2000 contient: 05Adresse 2001 contient: 03

Surveillance pendant l'exécution :

1. Ouvrir la fenêtre RAM
2. Noter les adresses à surveiller
3. Exécuter le programme (pas à pas ou continu)
4. Observer les changements en temps réel

Débogage :

- Vérifier où le programme stocke ses données
- Identifier écritures incorrectes
- Détecter corruption mémoire
- Tracer flux de données

Interprétation ASCII :

Exemple:Donnée: 48 65 6C 6C 6F 20 57 6F 72 6C 64 00ASCII: H e l l o W o r l d .Utile pour:- Chaînes de caractères- Messages de débogage- Identification rapide de données texte

6.3 Fenêtre Vue Mémoire - ROM

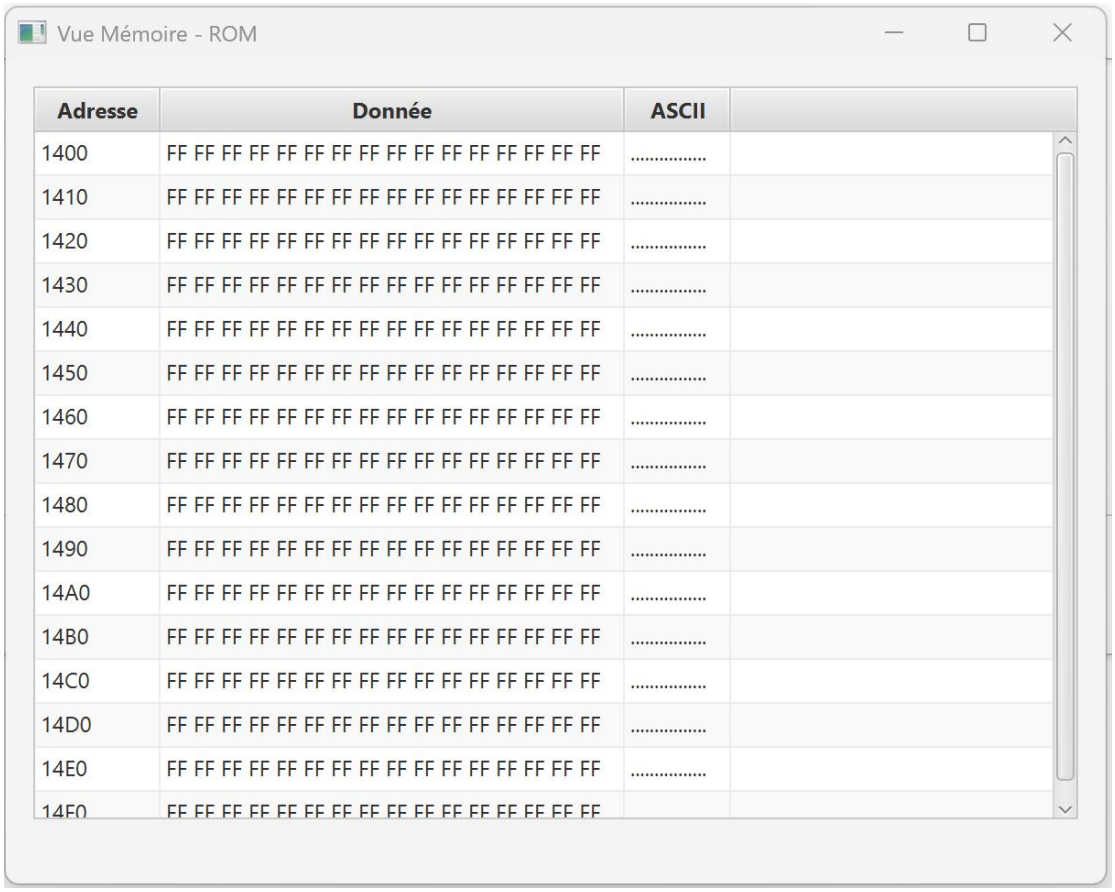


Figure 5 : Fenêtre Vue Mémoire - ROM

La fenêtre ROM affiche le contenu de la mémoire morte (Read-Only Memory) où est chargé le programme.

Caractéristiques d'Affichage

Format identique à RAM :

- 3 colonnes : Adresse | Donnée | ASCII
- 16 octets par ligne

Différence visuelle :

- **Bordure bleue distinctive** autour du tableau
- Indique zone protégée en écriture
- Adresses commencent typiquement à 1400

Exemple après assemblage :

Adresse	Donnée	ASCII
1400	86 05 C6 03 3D B7 20 00 3F FF FF FF FF FF FF FF	.....

1410 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....

Décodage du Code Machine

Correspondance opcode ↔ instruction :

- **86** : LDA immédiat
- **C6** : LDB immédiat
- **3D** : MUL
- **B7** : STA étendu
- **3F** : SWI

Initialisation et Contenu

État initial (avant assemblage) :

- Toutes les valeurs à **FF**
- Mémoire vierge
- Prête à recevoir le programme

Après assemblage :

- Remplie avec le code machine généré
- Commence à l'adresse ORG
- Contient opcodes + opérandes + données

Protection ROM :

- **Lecture seule** pendant l'exécution
- Impossible d'écrire dans cette zone avec STA/STB/STD
- Simule comportement ROM physique
- Protège le code contre corruption

Utilisation Pratique

Visualisation du code compilé :

1. Écrire le programme assembleur
2. Cliquer sur **Assembler**
3. Ouvrir la fenêtre ROM via bouton

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

7.4 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. Exemple Pratique

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

10.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.