

Simulation des circuits numériques et programmation en Assembleur 8086

**Réalisé par :
Nour ben Rhouma
Wiem Assadi**

Circuits numériques

Introduction

Ce projet porte sur deux domaines importants : les circuits combinatoires et la programmation en assembleur 8086. Dans la première partie, il s'agit de concevoir et de simuler des circuits comme des décodeurs, comparateurs et multiplexeurs à l'aide de logiciel SimulIDE, afin de mieux comprendre leur fonctionnement. La seconde partie consiste à écrire un programme en assembleur sur un émulateur 8086 pour manipuler et observer les registres (AX, BX, CX, DX) en utilisant des instructions simples comme MOV, ADD, SUB, INC et DEC.

1. Circuits combinatoires

1.1-Décodeurs 2x4

Un décodeur 2x4 est un circuit combinatoire utilisé pour activer une seule sortie parmi quatre possibles en fonction des combinaisons des deux entrées A et B

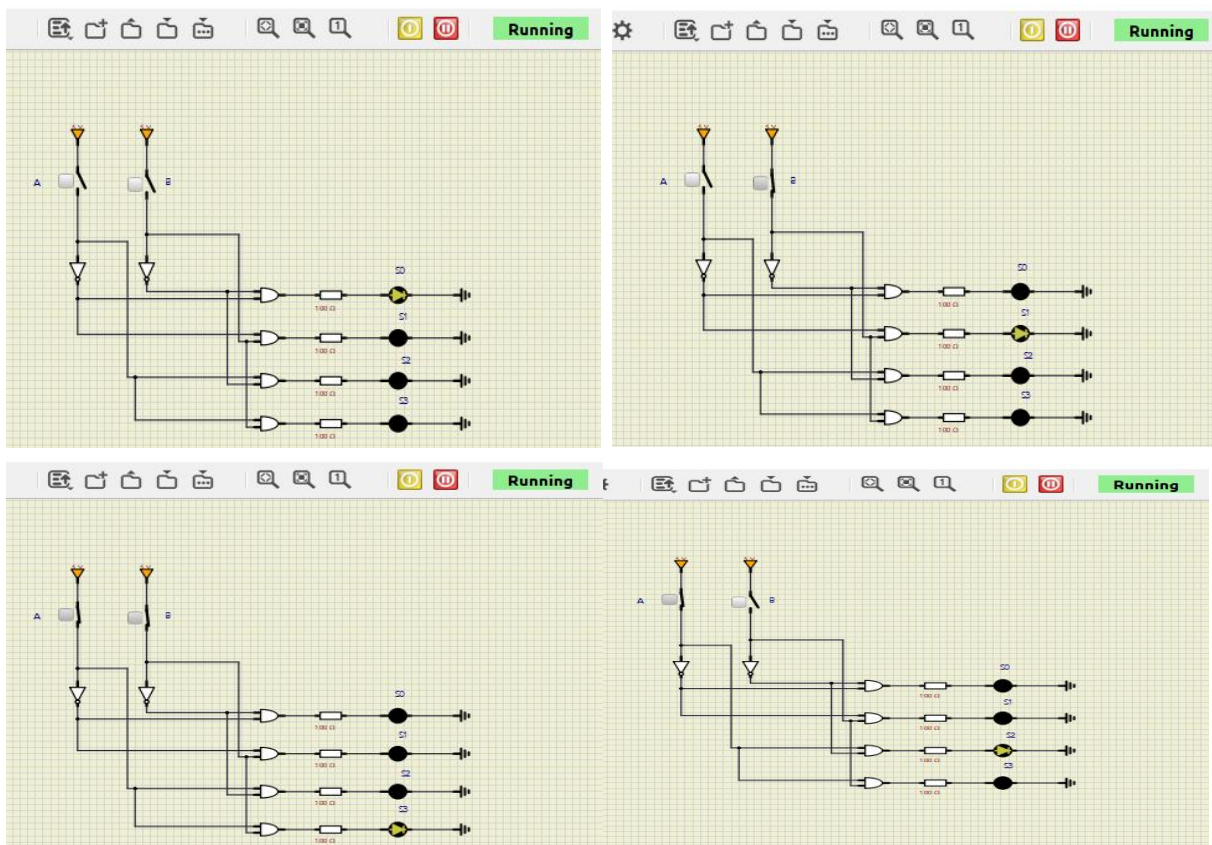


Figure 1 :logigramme de décodeur

1.2-composants nécessaires dans SimulIDE

Deux entrées binaires (A et B) : Ces entrées (interrupteurs) servent à sélectionner la sortie active.

Quatre sorties (S0, S1, S2, S3) : Seule une des sorties (Leds) sera activée pour chaque combinaison des entrées.

Des portes logiques (NOT, AND) : Utilisées pour générer les conditions nécessaires à l'activation des sorties.

Chaque sortie correspond à une équation logique bien définie :

$$S_0 = \overline{A} \cdot \overline{B}, \quad S_1 = \overline{A} \cdot B, \quad S_2 = A \cdot \overline{B}, \quad S_3 = A \cdot B$$

Résultat attendu

Après simulation, les résultats observés confirment que pour chaque combinaison des entrées une seule sortie est activée, suivant la table de vérité suivante :

A	B	S0	S1	S2	S3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Table1: Table de vérité du décodeur 2x4

2.1-Comparateur 1 bit

Le but de ce comparateur 1 bit est de comparer deux entrées binaires A et B, et d'afficher les résultats sur des LED en fonction de la comparaison.

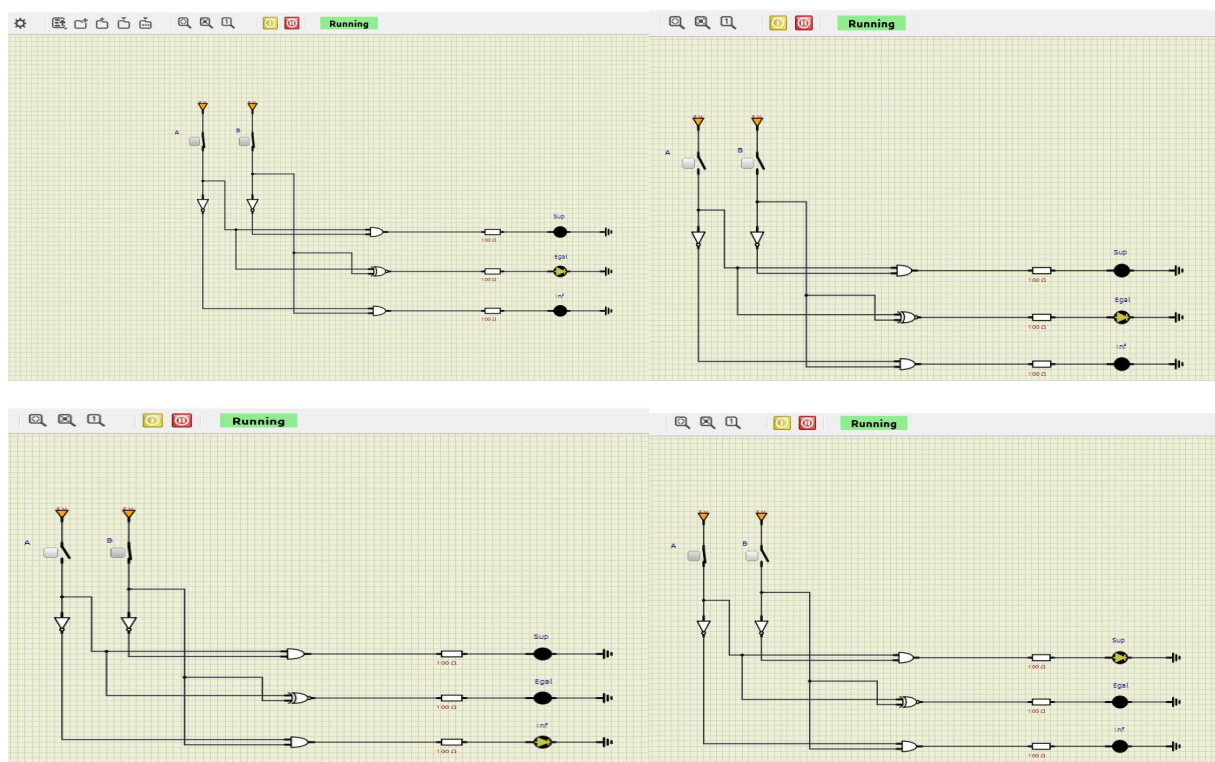


Figure 2: logigramme de comparateur 1 bit

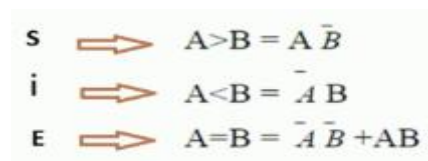
2.2-composants nécessaires dans SimulIDE

Deux entrées binaires (A et B) : Ces entrées (interpréteurs) servent à sélectionner la sortie active.

Trois sorties (Sup, Inf, Egal) : Seule une des sorties sera activée pour chaque combinaison des entrées.

Des portes logiques (NOT, AND, XOR) : Utilisées pour générer les conditions nécessaires à l'activation des sorties.

Chaque sortie correspond à une équation logique bien définie :


$$\begin{aligned} \text{S} &\Rightarrow A > B = A \bar{B} \\ \text{i} &\Rightarrow A < B = \bar{A} B \\ \text{E} &\Rightarrow A = B = \bar{A} \bar{B} + AB \end{aligned}$$

Étapes de simulation :

1. Configurer deux interrupteurs pour représenter les entrées A et B.
2. Relier les interrupteurs aux portes logiques pour implémenter les trois équations.
3. Connecter les sorties des portes logiques à trois LEDs pour visualiser les états.
4. Tester les quatre combinaisons possibles des entrées (A, B) et vérifier les résultats obtenus.

Résultat attendu

Après simulation, les résultats observés confirment que pour chaque combinaison des entrées une seule sortie est activée, suivant la table de vérité suivante :

A	B	Sup (A > B)	Inf (A < B)	Égal (A = B)
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Table2:Table de vérité du comparateur 1 bit

3.1multiplexeur (MUX) 8x1

Un **multiplexeur 8x1 (MUX)** est un circuit combinatoire qui sélectionne une entrée parmi **8 (D0 à D7)**, en fonction des combinaisons de **3 lignes de sélection (S0, S1, S2)**, et transmet cette entrée vers une sortie unique **Y**.

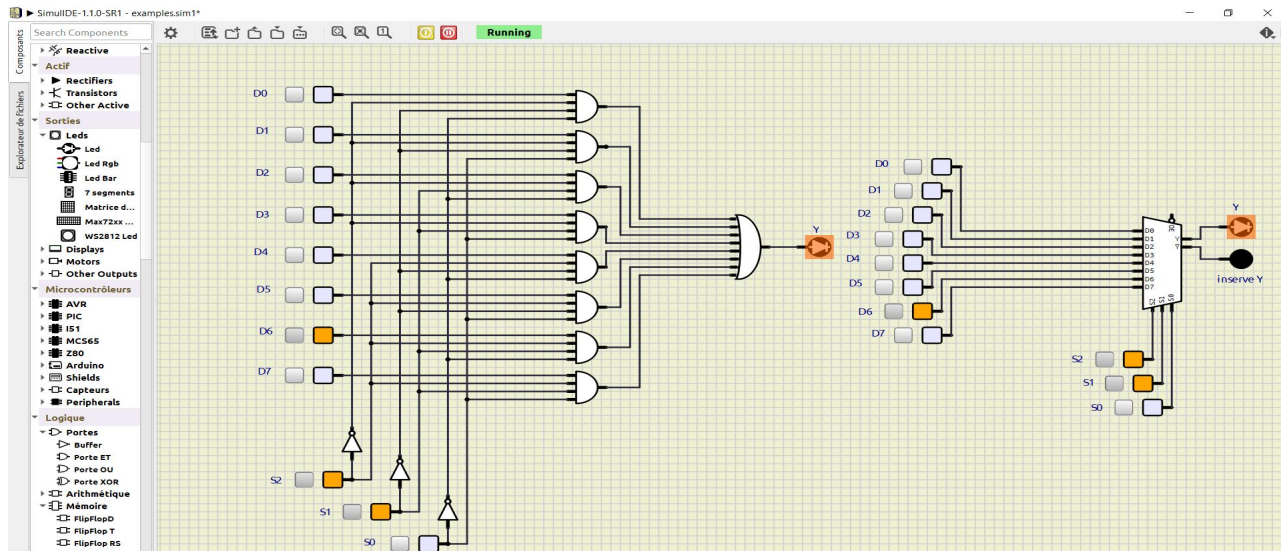


Figure 3 :logigramme de Multiplexeur 8x1

3.2-composants nécessaires dans SimulIDE

- **Entrées** : 8 interrupteurs (D0 à D7).
- **Portes logiques** : NOT, AND, et OR pour implémenter la formule logique.
- **Lignes de selection**: Trois interrupters (S0, S1, S2).
- **Portes logiques**
 - NOT: Inverse les sélecteurs si nécessaire.
 - AND: Combine les sélecteurs et les entrées.
 - OR: Regroupe toutes les sorties des portes AND.
- **LED**: Visualise la sortie(Y).

Étapes principales :

1. Configure les entrées (D0 à D7) et les lignes de sélection (S0, S1, S2).
2. Connecte les portes logiques (NOT, AND, OR) selon la formule logique :

$$Y = (\overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_0) + (\overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot D_1) + (\overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot D_2) + (\overline{S_2} \cdot S_1 \cdot S_0 \cdot D_3) \\ + (S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_4) + (S_2 \cdot \overline{S_1} \cdot S_0 \cdot D_5) + (S_2 \cdot S_1 \cdot \overline{S_0} \cdot D_6) + (S_2 \cdot S_1 \cdot S_0 \cdot D_7)$$

3. Relie la sortie de la porte OR à une LED pour afficher l'état de la sortie.
4. Teste les combinaisons des sélecteurs et vérifie que seule l'entrée sélectionnée passe à la sortie.

Résultat attendu :

Pour chaque combinaison de (S0, S1, S2), une seule entrée (D0 à D7) sera activée à la sortie Y, allumant la LED

.Les résultats sont résumés dans le tableau suivant :

S2	S1	S0	Y
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

Table3:Table de vérité du multiplexeur 8x1

2.Emulateur 8086

2.1 MOV

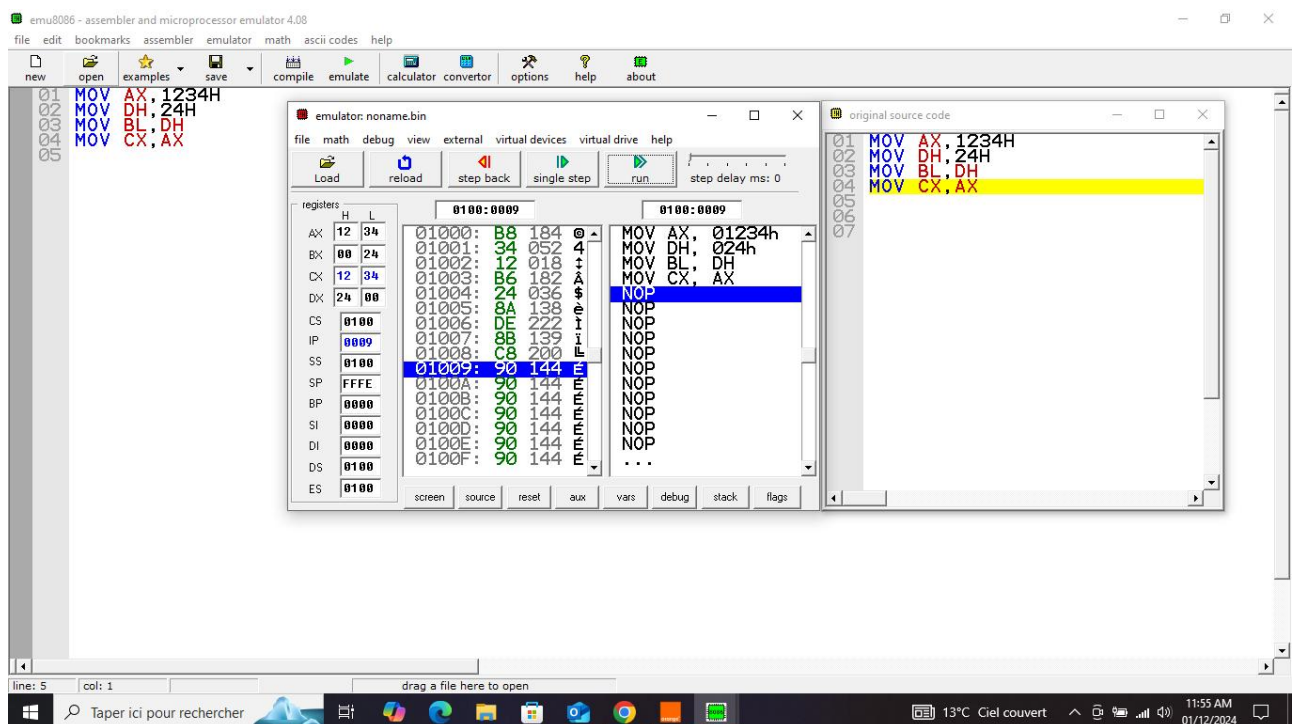


Figure4:transferts de données avec MOV

Ce programme utilise l'instruction **MOV** pour réaliser des transferts de données simples et permet d'observer l'état des différents registres dans un émulateur 8086 :

- Charge 1234H dans le registre AX.
- Charge 24H dans le registre DH.
- Copie la valeur de DH dans BL.
- Copie la valeur de AX dans CX.

2.2 ADD

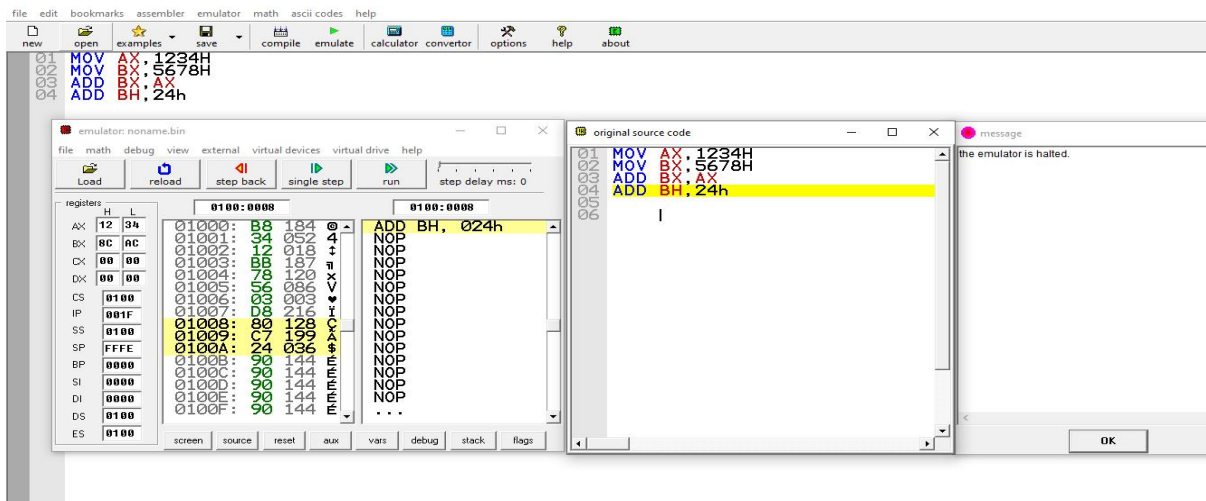


Figure5:Addition avec ADD

Ce programme démontre l'utilisation des instructions **MOV** pour charger des valeurs dans les registres et **ADD** pour effectuer des additions. Il ajoute la valeur de **AX** à **BX** et incrémente l'octet supérieur de **BX** (**BH**) de **24H**, permettant d'observer les modifications des registres après des opérations arithmétiques.

2.3 SUB

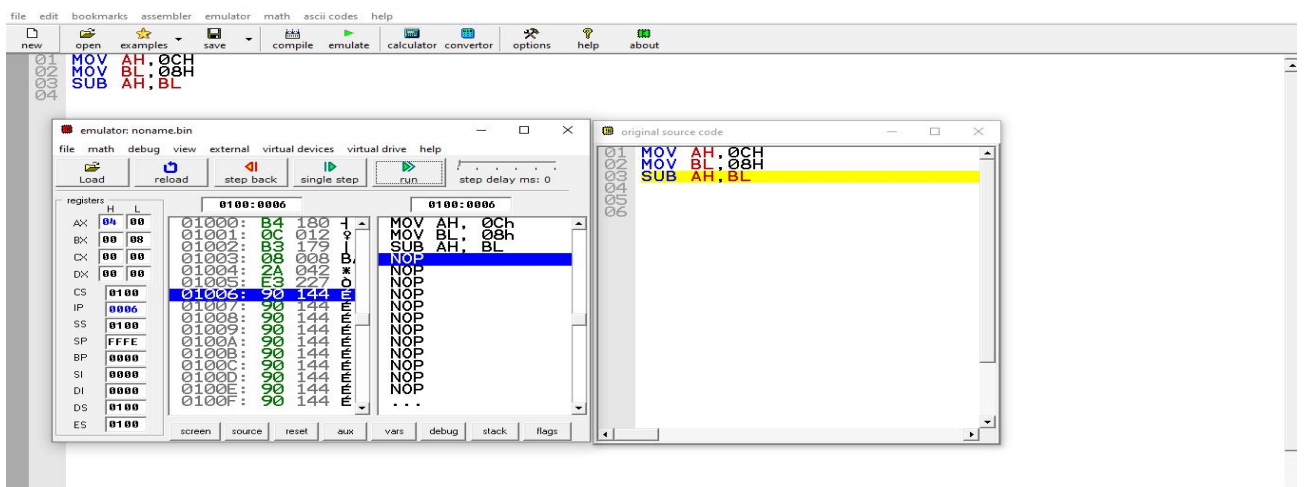


figure6:Soustraction avec SUB

Ce programme illustre l'utilisation de l'instruction **MOV** pour charger des valeurs dans des registres et de l'instruction **SUB** pour effectuer une soustraction. Il soustrait la valeur de **BL** (**08H**) du registre **AH** (**0CH**) et met à jour le contenu de **AH** avec le résultat.