TP Assembleur x86

Calculatrice Simple

PAUL-BASTHYLLE MASSE MASSE

GSI 531 - Architecture des Ordinateurs

Décembre 2025

Dépôt GitHub : https://github.com/massepaul19/Tp-No-1-assembleur

Table des matières

1 Introduction

Ce rapport présente la réalisation d'une calculatrice simple en assembleur x86 (architecture 32 bits). Le projet a été développé progressivement en 6 phases, permettant d'acquérir les compétences fondamentales en programmation assembleur sous Linux.

1.1 Objectifs pédagogiques

— Maîtriser les appels système Linux (sys_read, sys_write, sys_exit)

— Comprendre la manipulation des registres x86

— Apprendre les instructions arithmétiques (ADD, SUB, MUL, DIV)

— Utiliser les branchements conditionnels (CMP, JE, JNE)

— Développer une architecture modulaire professionnelle

1.2 Méthodologie de travail

Approche transparente : Combinaison de compétences et d'outils

Pour la réalisation de ce projet, j'ai adopté une méthodologie hybride combinant mes compétences existantes et l'utilisation d'outils modernes :

Mes compétences mobilisées :

— Maîtrise de la programmation modulaire (acquise en C)

— Connaissance des Makefiles pour l'automatisation

— Utilisation de Git pour le versionnage du code

— Capacité d'organisation et de structuration de projet

Utilisation de l'IA comme assistant :

— Apprentissage accéléré des concepts d'assembleur x86

— Aide au débogage des erreurs de compilation et d'exécution

— Amélioration de la documentation (README.md structuré)

— Suggestions pour l'organisation modulaire du code

Support de cours utilisé :

— Document `cours_assembleur_support_1.pdf`

— Énoncé du TP : `GSI_531_TP1_Assembleur_2025_2026_2.pdf`

Ma démarche : J'ai combiné mes connaissances en programmation (modularité, Makefile, Git) avec l'assistance de l'IA pour réaliser ce projet. L'IA m'a aidé dans l'apprentissage de l'assembleur x86 et dans le débogage, tandis que j'ai appliqué mes compétences pour structurer le projet de manière professionnelle.

2 Architecture du projet

2.1 Structure des fichiers

Le projet est organisé selon une architecture professionnelle :

Listing 1 – Arborescence du projet

```
Masse_paul_gsi_tp_assembleur/
          bin/                    # Executables generes
          code/                   # Fichiers source
```

```
                calco/              # Projet modulaire
                      Fonctions/      # Modules des operations
                            addition.asm
                            soustraction.asm
                            multiplication.asm
                            division.asm
                            menu.asm
                      main.asm        # Point d'entree
                phase1.asm          # Phases individuelles
                phase2.asm
                phase3.asm
                phase4.asm
                phase5.asm
          obj/                      # Fichiers objets
                calco/
          Makefile                # Automatisation
          README.md               # Documentation
          Reponses_questions.pdf  # Reponses QCM
```

2.2 Compilation avec Makefile

Un Makefile complet a été créé pour automatiser la compilation et les tests. Pour afficher toutes les commandes disponibles, commencer par taper :

```
make help
```

Voici la sortie complète du menu d'aide :

```
+============================================================+
|          TP ASSEMBLEUR x86 - CALCULATRICE SIMPLE           |
+============================================================+

  COMPILATION MODULAIRE :
  make all            - Compiler la calculatrice modulaire
                        et toutes les phases
  make calc           - Compiler la calculatrice modulaire

  PHASES INDIVIDUELLES :
  make phase1         - Compiler la phase 1 (affichage message)
  make phase2         - Compiler la phase 2 (lecture clavier)
  make phase3         - Compiler la phase 3 (addition simple)
  make phase4         - Compiler la phase 4 (menu + CMP/JE)
  make phase5         - Compiler la phase 5 (MUL/DIV)

  EXECUTION :
  make run            - Executer la calculatrice modulaire
  make run_phase1     - Executer la phase 1
  make run_phase2     - Executer la phase 2
  make run_phase3     - Executer la phase 3
  make run_phase4     - Executer la phase 4
  make run_phase5     - Executer la phase 5

  TESTS RAPIDES (Compile + Execute) :
  make test_phase1    - Compiler + executer phase 1
  make test_phase2    - Compiler + executer phase 2
  make test_phase3    - Compiler + executer phase 3
  make test_phase4    - Compiler + executer phase 4
  make test_phase5    - Compiler + executer phase 5
  make test_all       - Compiler + executer calculatrice complete

  NETTOYAGE :
```

```
  make clean          - Supprimer fichiers .o et executables
  make cleanall       - Supprimer obj/ et bin/ completement
  make efface         - Effacer le terminal

+============================================================+
|Auteur: PAUL-BASTHYLLE MASSE MASSE | GSI 531 | Decembre 2025|
+============================================================+
```

3 Développement progressif

3.1 Phase 1 : Affichage d'un message (2 points)

Objectif : Écrire un programme affichant "Bonjour, monde !" puis se terminant proprement.
Concepts abordés :

- Structure d'un programme assembleur (sections .data, .bss, .text)
- Appel système sys_write (EAX = 4)
- Appel système sys_exit (EAX = 1)

Compilation :

```
make phase1
./bin/phase1
```

Résultat : Programme fonctionnel affichant le message attendu.

3.2 Phase 2 : Lecture clavier (3 points)

Objectif : Lire un caractère depuis stdin et l'afficher.
Concepts abordés :

- Appel système sys_read (EAX = 3)
- Utilisation de buffers en section .bss
- Gestion des entrées utilisateur

Résultat : Lecture et affichage corrects du caractère saisi.

3.3 Phase 3 : Addition simple (3 points)

Objectif : Lire deux chiffres (0-9) et afficher leur somme.
Concepts abordés :

- Conversion ASCII → numérique : `sub al, '0'`
- Instruction ADD
- Conversion numérique → ASCII : `add al, '0'`

Exemple d'exécution :

```
Premier chiffre (0-9) : 5
Deuxieme chiffre (0-9) : 3
Resultat : 8
```

Résultat : Addition fonctionnelle pour les chiffres 0-9.

3.4 Phase 4 : Menu et branchements (4 points)

Objectif : Afficher un menu et exécuter une action selon le choix.
Concepts abordés :

— Instruction CMP (comparaison)
— Sauts conditionnels : JE (jump if equal), JNE (jump if not equal)
— Structure de contrôle (if/else en assembleur)

Résultat : Menu interactif avec branchements corrects.

3.5 Phase 5 : Multiplication et division (4 points)

Objectif : Utiliser les instructions MUL et DIV.
Concepts abordés :

— **MUL** : Résultat dans AX (AL × opérande)
— **DIV** : Quotient dans AL, reste dans AH
— Nettoyage du registre AH avant division

Exemple MUL :

```
6 x 7 = 42
AL = 6, operande = 7
Apres MUL : AX = 42
```

Exemple DIV :

```
17 / 5 = 3 reste 2
AX = 17, operande = 5
Apres DIV : AL = 3, AH = 2
```

Résultat : Multiplication et division fonctionnelles.

3.6 Phase 6 : Architecture modulaire (4 points)

Objectif : Créer une architecture professionnelle avec séparation des modules.
Architecture modulaire :

— **main.asm** : Point d'entrée, boucle principale
— **menu.asm** : Affichage du menu
— **addition.asm** : Fonction d'addition
— **soustraction.asm** : Fonction de soustraction
— **multiplication.asm** : Fonction de multiplication
— **division.asm** : Fonction de division

Directives de liaison :

— `global` : Exporte un symbole
— `extern` : Importe un symbole

Avantages :

— Code modulaire et réutilisable
— Maintenance facilitée
— Débogage simplifié
— Architecture professionnelle

Compilation multi-fichiers :

```
make calc
./bin/calc
```

Résultat : Calculatrice modulaire complète et fonctionnelle.

4 Concepts techniques maîtrisés

4.1 Registres x86 utilisés

Registre	Taille	Usage
EAX	32 bits	Numéro d'appel système, résultats
EBX	32 bits	1er argument (file descriptor)
ECX	32 bits	2ème argument (adresse buffer)
EDX	32 bits	3ème argument (longueur)
AX	16 bits	Résultat MUL, dividende DIV
AL	8 bits	Calculs arithmétiques, quotient DIV
AH	8 bits	Reste de DIV

TABLE 1 – Registres x86 utilisés dans le projet

4.2 Appels système Linux

Syscall	EAX	Description
sys_exit	1	Terminer le programme
sys_read	3	Lire depuis stdin
sys_write	4	Écrire vers stdout

TABLE 2 – Appels système utilisés

4.3 Instructions arithmétiques

— **ADD dest, src** : Addition (dest = dest + src)

— **SUB dest, src** : Soustraction (dest = dest - src)

— **MUL src** : Multiplication non signée (AX = AL × src)

— **DIV src** : Division non signée (AL = quotient, AH = reste)

5 Tests et validation

5.1 Procédure de test

Chaque phase a été testée individuellement avec la commande :

```
make test_phase1  # Compile et execute phase 1
make test_phase2  # Compile et execute phase 2
# ... etc
make test_all     # Teste la calculatrice complete
```

5.2 Résultats des tests

6 Conclusion

Ce projet m'a permis de :

1. **Maîtriser les fondamentaux** de l'assembleur x86 32 bits
2. **Comprendre les mécanismes bas niveau** : registres, appels système, pile

Phase	Statut	Commentaire
Phase 1		Affichage correct
Phase 2		Lecture/écriture OK
Phase 3		Addition fonctionnelle
Phase 4		Menu et branchements OK
Phase 5		MUL/DIV corrects
Phase 6		Architecture modulaire complète

TABLE 3 – Résultats des tests par phase

3. **Développer une architecture professionnelle** avec séparation modulaire
4. **Automatiser la compilation** avec un Makefile complet
5. **Utiliser l'IA comme outil pédagogique** pour accélérer l'apprentissage

6.1 Points forts du projet

— Architecture modulaire professionnelle
— Documentation complète (README.md détaillé)
— Makefile avec automatisation complète
— Code commenté et structuré
— Toutes les phases fonctionnelles
— Dépôt GitHub avec historique de commits

6.2 Difficultés rencontrées

— Compréhension initiale des registres et de leur hiérarchie (EAX/AX/AH/AL)
— Gestion correcte de la conversion ASCII numérique
— Nettoyage du registre AH avant division
— Compilation multi-fichiers avec directives global/extern

6.3 Compétences acquises

Ce TP m'a permis d'acquérir des compétences solides en :

— Programmation système bas niveau
— Débogage en assembleur
— Architecture des processeurs x86
— Méthodologie de développement modulaire

Projet complet disponible sur :
https://github.com/massepaul19/Tp-No-l-assembleur