INFO-F105 – Langages de programmation 1
Projet – Phase 1

1 Introduction

Une entreprise qui fabrique des consoles de jeux souhaite intégrer un nouveau processeur dans leurs consoles, et vous êtes en charge de sa conception.

Avec ce projet, vous démontrerez votre compréhension du langage C++ tout en vous familiarisant avec le modèle d'exécution d'un processeur tel que vu au cours.

2 Consignes

Vous devez implémenter un programme en C++ qui exécute des instructions lues **à la volée** depuis un fichier texte dont le chemin est passé en argument à votre programme.

2.1 Registre

Lors de la première phase, le jeu d'instructions supporté par votre processeur permet uniquement des opérations sur un unique registre de travail **initialisé à 0**. Vous devriez donc avoir une variable numérique mise à jour à chaque instruction en fonction de l'opération demandée.

Ce registre contient un entier **non signé** et **saturé** de **16 bits**.

Un nombre saturé est ramené à ses bornes lorsqu'elles sont dépassées. En d'autres termes :

$$\text{saturated}(n) = \min(\max(n, 0), 2^{16})$$

Vous devrez donc tenir compte de cela dans les opérations d'addition et de soustraction.

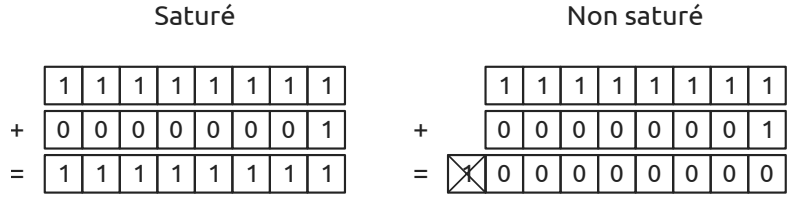

Figure 1: Addition de `uint8_t` saturés vs `uint8_t` classiques

2.2 Instructions

Chaque instruction est composée d'un opcode et de 0 ou 1 opérande numérique.

Le tableau ci-dessous reprend les instructions que votre processeur doit supporter.

Instruction	Description
`SET n`	Définit la valeur du registre à `n`
`ADD n`	Ajoute `n` à la valeur du registre
`SUB n`	Soustrait `n` à la valeur du registre
`PRINT`	Affiche la valeur du registre
`IFNZ`	Ignore l'instruction suivante si la valeur du registre est `0`

2.3 Structure

Pour vous guider, voici 3 fonctions que vous devriez implémenter en plus du `main` :

```
// Retourne l'instruction sans les opérandes
std::string parse_opcode(const std::string& instr);

// Retourne l'opérande (paramètre) sous forme numérique
uint16_t parse_operand(const std::string& instr);

// Exécute le programme dans le fichier texte 'program_path'
void exec(const std::string& program_path);
```

3 Exemple

Le fichier d'entrée contient une instruction par ligne, comme dans l'exemple ci-dessous.

```
SET 5
ADD 6
PRINT
SUB 4
IFNZ          →    11
PRINT              7
SUB 10
IFNZ
PRINT
```

Pour lancer les tests sur votre programme :

```
python3 tests1.py chemin/vers/votre/executable
```

N'hésitez pas à créer une entrée `test` dans votre `Makefile` pour vous simplifier la tâche.

4 Rapport

Outre le code, il vous est demandé de remettre un rapport de maximum 1 page qui développera très brièvement l'organisation de votre code et vos choix d'implémentation.

Vous y décrirez notamment comment vous avez implémenté votre registre saturé.

5 Remise

Avant de remettre votre travail, veillez à bien respecter les points suivants :

1. Le code doit être **commenté**.
2. Le code doit compiler sans **warning**, à moins qu'un commentaire ne justifie sa présence.
3. Les **cas limites** doivent être pris en compte, notamment la saturation du registre.
4. Le programme doit passer tous les **tests** du fichier `tests1.py`
5. Tous les fichiers sources (`.cpp`, `.hpp`) et `Makefile` doivent être remis.

Consignes de remise

1. Mettez le tout dans un fichier `<matricule>.zip`
2. Remise sur l'UV
3. Date limite : le **vendredi 14 mars** avant **22h**