INFO-F105 – Langages de programmation 1
Projet – Phase 2

1 Introduction

Lors de cette deuxième phase, vous implémenterez la mémoire utilisée par votre processeur ainsi que les opérations nécessaires à son utilisation. Vous ajouterez également de nouveaux registres pour permettre plus de flexibilité, mais aussi un stack et de nouvelles instructions.

2 Consignes

2.1 Registres

Pour cette seconde phase, vous devez adapter les instructions de la phase précédente pour permettre l'utilisation de 4 registres différents : `a`, `b`, `c` et `d`.

Ces registres contiennent des entiers non signés et saturés de 16 bits et sont initialisés à 0.

Voici un aperçu des instructions à modifier :

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

Certaines instructions doivent désormais également permettre l'addition de registres.

Instruction	Description
`SET r1 r2`	Définit la valeur de `r1` à celle de `r2`
`ADD r1 r2`	Ajoute la valeur de `r2` à celle de `r1`
`SUB r1 r2`	Soustrait la valeur de `r2` à celle de `r1`

2.2 Mémoire

La mémoire consiste en un tableau de bytes (`uint8_t`) de taille 2^B, où `B` est une valeur constante indiquant la taille des adresses que l'on fixe à 8 bits. Ce tableau doit être défini dans le fichier `memory.cpp`, de même que toutes les autres fonctionnalités liées à la mémoire.

Implémentez les opérations de lecture (`read`) et d'écriture (`write`) dans la mémoire.

```
uint16_t read(uint8_t address);
void write(uint8_t address, uint16_t value);
```

N'oubliez pas que vos registres contiennent toujours des valeurs sur 16 bits. **Hint** : *boutisme*

Stack

La mémoire de votre processeur est (conceptuellement) segmentée en deux parties de 16 et 240 bytes respectivement, comme représenté sur la figure 1.

Le stack (= *pile*) est une zone mémoire un peu particulière qui supporte deux opérations :

```
// Ajoute une valeur au sommet du stack
void push(uint16_t value);

// Retire la valeur au sommet du stack et retourne cette valeur
uint16_t pop();
```

Ces opérations impliquent de mettre à jour un registre `SP` (Stack Pointer) qui contient l'adresse (indice) du sommet du stack et permet de savoir où ajouter de nouvelles valeurs.

→ Implémentez ces opérations en tenant compte des cas limites.

Attention : le stack ne peut pas déborder sur la deuxième partie de la mémoire. Si tel est le cas, vous devez quitter le programme en utilisant `std::exit(1)`.

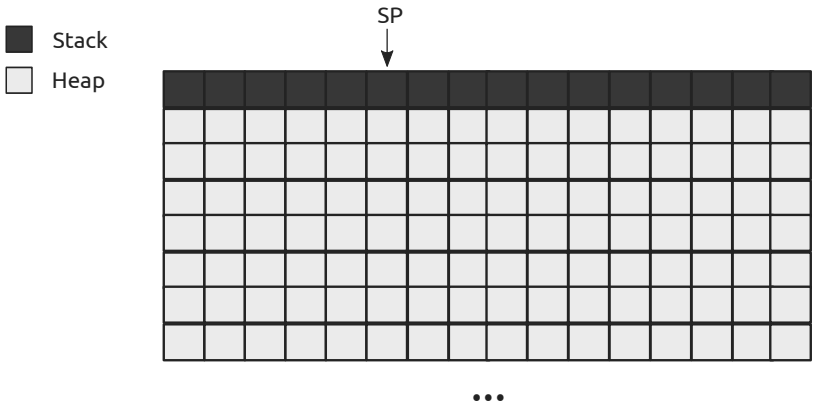

Figure 1: Segmentation de la mémoire

2.3 Instructions

Le tableau ci-dessous reprend les nouvelles instructions à ajouter dans votre processeur.

Instruction	Description
`STORE a r`	Enregistre la valeur de `r` à l'adresse `a`
`LOAD a r`	Charge la donnée à l'adresse `a` dans `r`
`PUSH r`	Ajoute la valeur de `r` au stack
`POP r`	Retire une valeur du stack et la met dans `r`

3 Exemple

```
SET a 150
SET b 130
PUSH a
ADD a b
PUSH a
STORE 100 a
STORE 101 b
POP c
PRINT c
POP d
PRINT d
LOAD 100 b
IFNZ b
PRINT b
LOAD 101 a
PRINT a
SUB b b
IFNZ b
PRINT b
```

$\rightarrow$

```
280          280
150   OU     150
256          33304
130          130
```

Selon votre implémentation, vous devriez obtenir l'un ou l'autre de ces résultats. Prenez bien le temps de comprendre pourquoi vous arrivez à ce résultat en utilisant votre debugger.

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 préciserez notamment quel résultat vous obtenez avec l'exemple ci-dessus et pourquoi.

5 Remise

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

1. La **librairie standard** est la seule librairie autorisée.
2. Le code doit être **commenté**.
3. Le code doit compiler sans **warning**, à moins qu'un commentaire ne justifie sa présence.
4. Les **cas limites** doivent être pris en compte, notamment les bornes du stack.
5. Le programme doit passer tous les **tests** du fichier `tests2.py`
6. 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 11 avril** avant **22h**