ARCHITECTURE DES ORDINATEURS

TD : **03**

Appels de fonction en y86

Rappels

- L'appel de fonctions en y86, comme en x86, repose sur un certain nombre de conventions.
- La valeur de retour d'une fonction est contenue dans le registre `%eax`.
- Les paramètres sont passés sur la pile. Ils sont donc empilés par l'appelant avant d'effectuer `call`, et utilisés sur place par l'appelé. C'est l'appelant qui s'occupe de les enlever de la pile.
- Concernant l'utilisation et la sauvegarde des registres lors des appels de fonction, on distingue deux types de registres :
  - registres « *caller save* » (« *sauvegardés par l'appelant* ») : ces registres sont librement utilisables par la fonction appelée, et doivent être sauvegardés si nécessaire par la fonction appelante. Ce sont les registres `%eax`, `%ecx` et `%edx`,
  - registres « *callee save* » (« *sauvegardés par l'appelé* ») : ces registres doivent être restitués intacts à la fonction appelante. Toute fonction désireuse de les utiliser doit en sauvegarder la valeur puis la restaurer avant de terminer. Ce sont les registres `%ebx`, `%esi`, `%edi` et `%ebp`.

  Le registre `%esp` est à part, car il est directement impliqué dans l'appel de fonction.

Exercice 1 : Appel de fonction avec paramètres

Réalisez l'équivalent du code C suivant :

```
long f (long i, long j) {
  return (i + j);
}

long a;

void main1 () {
  a = f (1, 2);
  halt();
}

void main2 () {
  a = f (1, 2) + f (3, 4);
  halt();
}

long g (long i, long j) {
  return (f (i, j);
}

# exo à la maison
void main3 () {
```

```
  a = f (1, f (1, 2));
  halt();
}

void main4 () {
  a = f (f (1, 2), f (3, 4));
  halt();
}

long h (long i, long j) {
  return (f (i, j) + f (j, i));
}

void main4 () {
  a = g (1, 2);
  halt();
}
```

Rappels

- Les variables locales sont définies dans la pile après l'adresse de retour.
- Lorsqu'une fonction nécessite des variables locales, on décrémente `%esp` de la taille nécessaire, et l'on peut accéder aux variable en partant de `%esp`. Attention par contre, l'accès aux paramètres de fonction depuis `%esp` est décalé, du coup.

Exercice 2 : Fonction avec variable locale

Modifiez le code de l'exercice précédent pour réécrire la fonction `f` afin qu'elle mette en œuvre une variable locale, selon le code C suivant. `h` est une fonction que l'on suppose être déjà définie ailleurs.

```
long f (long i, long j) {
  long x;
  x = i + j;
  ...
  x += h(j)
  ...
  return (x);
}
```