Université de Bordeaux
Licence STS

université de BORDEAUX

ARCHITECTURE DES ORDINATEURS

TD : **04**

Utilisation des paramètres en y86

Exercice 1 : Passage par référence et adresses

Question 1

Réalisez l'équivalent du code C suivant :

```
void f (long * x, long y) {
  *x = y;
}

long t, u = 2;

void main () {
  f (&t, u);
  halt ();
}
```

Question 2

Que se passerait-il si l'on appelait `f (t, u)` au lieu de `f (&t, u)` ?

Exercice 2 : Parcours de tableau

Réalisez l'équivalent du code C suivant :

```
void f (long n, long * p) {
  for ( ; n > 0; p ++, n --)
    (*p) ++;
}

long t[4] = { 1, 2, 3, 4 };
void main () {
  f (4, t);
  halt ();
}
```

Attention, en C le tableau `t` est passé par référence ; il n'est pas copié sur la pile. Pour des raisons d'efficacité, n'utilisez pas d'emplacement mémoire pour le pointeur `p`, mais utilisez le registre `edx` à la place.

Exercice 3 : Liste d'arguments variable

Écrivez en assembleur une fonction calculant la somme de ses arguments `somme (n, ...)`, où `n` est le nombre d'arguments à additionner.

Pour aller plus loin...

Exercice 4 : Tout ensemble

Réalisez l'équivalent du code C suivant :

```
long g (long i) {
  return (i + 1);
}

void f (long t[n], long n) {
  long i;
  for (i = 0; i < n; i ++)
    t[i] = g (t[i]);
}

long t[] = {3, 5, 2, 6};

void main () {
  f (t, 4);
  halt ();
}
```

Exercice 5 : Multiplication

Réalisez en assembleur une fonction `long mult (long i, long j)`. Puisque l'on ne dispose pas d'instruction `mult`, il s'agit donc de le faire à la main, en ajoutant par exemple `j` à un accumulateur `i` fois (on ne traitera pas le cas des nombres négatifs).

Quelle est la complexité de cette fonction ?

Exercice 6 : Multiplication rapide

Observons maintenant avec la notation binaire $i = i_{31}i_{30}...i_1i_{0(2)} = i_{31}.2^{31} + i_{30}.2^{30} + ...i_1.2^1 + i_0.2^0$ (on ne traitera pas le cas des nombres négatifs) que

$$i \times j = i_{31}.j.2^{31} + i_{30}.j.2^{30} + ...i_1.j.2^1 + i_0.j.2^0$$

et donc il suffit de calculer les $j.2^n$ et les i_n pour n compris entre 0 et 31, et d'ajouter $j.2^n$ à l'accumulateur si i_n est égal à 1. Pour optenir successivement les $j.2^n$, il suffit d'ajouter j à lui-même et d'itérer. Pour les i_n, il suffit d'utiliser `& 1` en C (`iandl 1, %eax` en assembleur) pour extraire le bit de poids 0, et d'utiliser `>> 1` en C (`shrl %eax` en assembleur) pour décaler d'un bit vers la droite, et d'itérer. Écrivez la version C, puis la version assembleur.

Quelle est la complexité de cette fonction ?