N7 - $1^{ère}$ année SDN - T.P. n^o 3-4 d'architecture des ordinateurs
Processeur CRAPS : Programmation en langage d'assemblage

Somme des éléments d'un tableau

Ecrire un programme qui calcule la somme des éléments d'un tableau T de n entiers. L'algorithme est le suivant.

> S $\leftarrow$ 0;
> **Pour** i **depuis** 0 **jusqu'à** $n-1$ **pas** 1 **Faire**
>   S $\leftarrow$ S + $T[i]$;
> **FinPour**;

Recherche du plus grand élément d'un tableau

Ecrire un prgramme qui recherche le plus grand élément dans un tableau T de N entiers. L'algorithme est le suivant.

> Max $\leftarrow$ $T[0]$;
> **Pour** i **depuis** 1 **jusqu'à** $n-1$ **pas** 1 **Faire**
>   **Si** $T[i] > Max$ **Alors**
>   S $\leftarrow$ $T[i]$;
>   **FinSi**;
> **FinPour**;

Tri à bulles

On va illustrer l'utilisation des sous-programmes sur le tri d'un tableau de n entiers en utilisant le tri à bulle, décrit par l'algorithme suivant.

> **Pour** i **depuis** $n-1$ **jusqu'à** 1 **pas** -1 **Faire**
>   **Pour** j **depuis** 0 **jusqu'à** $i-1$ **pas** 1 **Faire**
>   permuter (T, j);
>   **FinPour**;
> **FinPour**;

Le principe de cet algorithme est de placer la plus grande valeur dans la dernière case du tableau, puis la plus grande valeur restante dans l'avant dernière case, et ainsi de suite. La procédure $Permuter(t, j)$ permute les éléments $T[j]$ et $T[j+1]$ si $T[j] > T[j+1]$

Crible d'Erathostène sur un tableau d'entier

L'objectif est d'éliminer d'un tableau T de n entiers tous les éléments qui sont multiples d'un autre élément du tableau.
Par exemple, si le tableau T contient les éléments

20 13 15 3 12 23 16 7 4 37

On élimine les éléments 20 (multiple de 4), 15 (multiple de 3), 12 (multiple de 3) et 16 (multiple de 4). Il reste donc

$$13 \quad 3 \quad 23 \quad 7 \quad 4 \quad 37$$

L'ordre dans lequel sont les éléments du tableau à la fin n'a pas d'importance.
On se propose de procéder en deux étapes :

- tri du tableau (en utilisant le tri à bulles de l'exercice précédent),
- Elimination des multiples en utilisant un crible du type de celui d'Erathostène pour la recherche des nombres premiers.

L'algorithme du crible est le suivant.

Pour i **depuis** 1 **jusqu'à** $n-1$ **pas** 1 **Faire**
    Elim[i] $\leftarrow$ 0;
FinPour;
Pour i **depuis** 1 **jusqu'à** $n-2$ **pas** 1 **Faire**
    Si $Elim[i] = 0$ **Alors**
        x $\leftarrow$ T[i];
        Pour j **depuis** i + 1 **jusqu'à** $n-1$ **pas** 1 **Faire**
            TantQue $x < T[j]$ **Faire**
                $x \leftarrow x + T[i]$;
            FinTantQue;
            Si $x = T[j]$ **Alors**
                $Elim[j] \leftarrow 1$;
            FinSi;
        FinPour;
    FinSi;
FinPour;

Evaluation d'une expression arithmétique en notation postfixée (polonaise inversée) à l'aide d'une pile

Il existe différentes manières d'écrire un expression arithmétique. La notation classique place l'opérateur entre les opérandes. Elle nécessite des parenthèses. Dans la notation polonaise inversée, l'opérateur est placé après les deux opérandes et les parenthèses deviennent inutiles. Par exemple, l'expression

$$(15 + (7 - 4)) - 2$$

en notation classique devient

$$15\ 7\ 4\ -\ +\ 2\ -$$

en notation polonaise inversée.
L'évaluation d'une expression en polonaise inversée s'effectue en utilisant une pile. Au départ, la pile est vide. Les termes sont lus de gauche à droite. Lorsque le terme lu est un entier (opérande), il est empilé. Lorsque le terme lu est un opérateur, on dépile les deux dernières valeurs empilées, on leur applique l'opérateur et on empile le résultat. A la fin, il doit y avoir une seule valeur dans la pile, qui correspond à la valeur de l'expression.
Chaque terme de l'expression est codée sur un mot mémoire de 32 bits, suivant le format suivant :

Type (1 bit)	Valeur (31 bits)

Lorsque $Type = 0$, le terme est un opérande. $Valeur$ est donc un entier signé sur 31 bits.
Lorsque $Type = 1$, le terme est un opérateur (+ lorsque $Valeur = 1$, $-$ lorsque $Valeur = 2$) ou le marqueur de fin de l'expression ($Valeur = 0$).
L'algorithme est le suivant.

```
i ← 0;
TantQue Expr[i] ≠ fin Faire
    Si Expr[i] est un opérande Alors
        Empiler (opérande);
    Sinon
        op2 ← dépiler ();
        op1 ← dépiler ();
        Si opérateur d'addition Alors
            Empiler (op1 + op2);
        Sinon
            Empiler (op1 − op2);
        SinSi;
    SinSi;
    i ← i + 1;
FinTantQue;
Res ← dépiler ();
```