TP 4 – Assembleur MIPS : Tas, pile, fonctions

Vous trouverez la liste complète des appels système disponibles avec `Mars` à l'adresse : `http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html`.

On veillera dès le début du TP à gérer la pile de fonction, et à respecter les conventions d'appels de fonction (en particulier, les registres temporaires sauvegardés `$si` seront sauvegardés si utilisés).

Téléchargez le fichier tp4src.s sur Moodle. Le fichier contient :
- les spécifications, entêtes et squelettes des fonctions à programmer pendant ce TP,
- un `main` avec une trame de test pour les fonctions à programmer,
- une fonction `AfficheEntier` permettant d'afficher un entier se trouvant dans le registre `$a0`,
- une fonction `AfficheTableau` permettant d'afficher un tableau d'entier de taille `$a0` à l'adresse `$a1`.

Ouvrez le fichier tp4src.s dans le simulateur Mars (rappel pour exécuter le programme : «java -jar Mars4_5.jar»).

1 Allocation de place dans le tas

1 – Quel appel système permet de réserver de la place dans le tas du processus ? Complétez la fonction `CreerTableau` afin qu'elle réserve la place nécessaire dans le tas pour un tableau de `$a0` entiers. Le registre `$v0` recevra l'adresse du début de la zone allouée.

2 – Terminez de programmer la fonction `CreerTableau` afin qu'elle initialise le tableau alloué :
- si `$a1` contient 0, le tableau sera trié par ordre croissant (avec des entiers non tous nuls),
- si `$a1` contient 1, le tableau sera trié par ordre décroissant (avec des entiers non tous nuls),
- si `$a1` contient 2, le tableau sera initialisé avec des entiers aléatoires.

2 Place d'un entier dans un tableau trié

On veut rechercher la place que devrait prendre un entier dans un tableau trié, d'abord de manière itérative, puis de manière récursive.

3 – Complétez la fonction `CherchePlace` qui prend en entrée :
- la taille du tableau dans `$a0`,
- l'adresse du premier élément du tableau supposé trié dans `$a1`,
- l'entier dont on cherche la place dans `$a2`,

et qui renvoie dans `$v0` l'offset en octets depuis le début du tableau de la place à laquelle l'entier dans `$a2` devrait se trouver. Cette fonction sera programmée de manière itérative. Testez votre fonction avec un tableau trié !

4 – Complétez la fonction `CherchePlaceRec` avec les mêmes spécifications que `CherchePlace`, qui recherche de manière récursive la place de l'entier dans `$a2`. Testez votre fonction avec un petit tableau trié, et exécutez le processus pas à pas pour voir le contenu des registres `$ra`, `$a0`, `$a1`, `$a2`.

3 Tri par insertion

On veut à présent implémenter en langage assembleur MIPS le tri par insertion d'un tableau d'entiers.

5 – Complétez la fonction `Decalage` qui prend en entrée :
- la taille du tableau dans `$a0`,
- l'adresse du premier élément du tableau dans `$a1`,

et qui décale d'une case à droite chaque élément du tableau. On suppose que les quatres octets suivant le tableau passé à l'entrée de la fonction peuvent être écrits.

6 – Complétez la fonction `Inserer` qui prend en entrée :
- la taille du tableau dans `$a0`,
- l'adresse du premier élément du tableau dans `$a1`,
- l'entier à insérer dans `$a2`,

et qui insère dans le tableau à sa place l'entier contenu dans `$a2`. On suppose que les quatres octets suivant le tableau passé à l'entrée de la fonction peuvent être écrits.

7 – Complétez la fonction itérative `Tri` qui prend en entrée :

— la taille du tableau dans `$a0`,
— l'adresse du premier élément du tableau dans `$a1`,
et qui trie le tableau par insertion.

4 Tri par insertion récursif

8 – Programmez et testez la fonction récursive `TriRec` qui trie un tableau de taille `$a0` à l'adresse `$a1`.