Licence Informatique S3
Architecture des ordinateurs
Aix Marseille Université 2017-2018
`http://pageperso.lif.univ-mrs.fr/~peter.niebert/archi2016.php`

TP5 : projet programmation MIPS

Rendez ce projet noté dans la dernière séance à votre enseignant de TP avec un rapport et les sources. Rendez des réalisations partielles commentées, si vous n'arrivez pas au bout de certains exercices.

Dans ce projet, vous allez programmer en assembleur MIPS. Pour ceci, nous utilisons le simulateur MARS . Téléchargez la version 4.5 et lancez le simulateur de la ligne de commande avec `java -jar mars4_5.jar`.

Attention, activez l'option *delayed branching* pour avoir le comportement normal de MIPS, c'est-à-dire d'exécuter l'instruction qui suit un branchement ou un saut. Le simulateur MARS permet pour des raisons historiques de sa fonction pédagogique la simulation sans l'exécution de cette instruction ce qui ne correspond pas au vrai comportement de ces processeurs ni à ce que vous avez vu dans l'enseignement !

Exercice 1 [Observation] L'exemple ci-dessous illustre le mécanisme de MARS pour entrée et sortie en passant par l'instruction `syscall`. De tels appels du système (instruction `syscall`) sont en pratique cachés dans la profondeur des bibliothèques E/S pour permettre aux applications d'utilisateurs d'invoquer des services privilégiés (Kernel). Ici, ils servent pour interagir avec l'environnement du simulateur.

Ouvrez le programme affiche_int.s et assemblez-le. Exécutez-le pas à pas en observant la mémoire et les modifications des registres.

Exercice 2 [nombres de Fibonacci] La célèbre suite de Fibonacci est définie comme $f(0) = 0, f(1) = 1, f(n+2) = f(n) + f(n+1)$ ce qui donne pour la suite $f(0), f(1), f(2), ...$ les valeurs
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$

- Ecrire en C une itérative (non récursive) pour calculer le nombre $f(n)$. Indication : il suffit de garder en mémoire deux valeurs $f(k), f(k+1)$ pour calculer avec 3 affectations les valeurs $f(k+1), f(k+2)$. Une boucle `for` suffit alors pour calculer $f(n)$.
- Traduire votre fonction C en Assembler MIPS. Réaliser dans la fonction main une boucle pour afficher les valeurs $f(0)...f(100)$ pour montrer le fonctionnement de votre fonction. Afin de respecter les conventions d'utilisation de registres, vous pouvez utiliser dans main les registres `$s0 ...` et dans votre fonction `int fib(int n)` des registres `$t0...`.

Exercice 3 [Récursion générale et pile]

L'instruction `jal` met l'adresse de retour dans le registre $`ra`, un retour simple est alors possible avec `jr $ra`. Cependant, un appel récursif risque d'écraser $`ra`, ainsi que des variables tenues dans d'autres registres. Pour y remédier, il faut gérer la *pile* C, un tableau dynamique dans la mémoire accédée par `$sp`, le *stack pointer*, pointant sur le sommet de la pile. Pour ajouter une nouvelle donnée à la pile (push) on écrit dans la donnée à l'adresse `$sp - 4` et on décrémente $`sp` de 4. Pour supprimer une donnée en tête de la pile (pop), on incrémente `$sp` de 4. Il est plus efficace de décrémenter une fois `$sp` en début de fonction pour réserver de la place pour les sauvegardes et de le libérer cette espace avec le retour en incrémentant `$sp` avec le même montant.

Avant un appel récursif, on sauvegarde alors au moins la valeur de $`ra` sur la pile. Par convention, une pile croit vers le bas (adresses descendantes).

Pour illustration, essayez et lisez le programme fact.s, qui calcule la factorielle d'un nombre de façon récursive.

Sur le modèle de `fact.s`, écrivez une fonction `int fib(int x)` pour le calcul des nombres de Fibonacci et illustrez son fonctionnement en invoquant cette fonction à partir de `main` pour des paramètres $0 \leq x \leq 3$ et $0 \leq y \leq 10$.

```
int ack(int x,int y){
 if(x==0) return (y+1);
 if(y==0) return (ack(x-1,1));
 return (ack(x-1,ack(x,y-1)));
}
```

Exercice 4 [Tableaux] Vous pouvez saisir directement dans un fichier assembleur un tableau en mémoire (section `.data`!) en procédant comme dans l'exemple suivant :

```
list:  .word 3, 0, 1, 2, 6, -2, 4, 7, 3, 7
```

Le petit bout de code suivant illustre le calcul d'adresses et place la valeur de list[6] dans le registre $t4 :

```
la $t3, list          # met l'adresse de list dans $t3
li $t2, 6             # met l'indice dans $t2
sll $t2, $t2, 2       # t2= t2*4
add $t1, $t2, $t3     # calcule l'adresse de la case 6 (list+ 6x4)
lw $t4, 0($t1)        # stocke la valeur de list[6]
```

Transformez la fonction recursive quicksort décrite ici en assembler MIPS et testez la.

Exercice 5 [Memory mapped IO (bonus)] Ouvrez dans MARS dans le menu le *Keyboard and Display MMIO Simulator*, puis appuyez sur le bouton *Help*. Il explique comment accéder à l'affichage et au clavier via les adresses `0xffff0000` ... `0xffff000c`.

— Utilisez cette interface pour afficher les lettres `a` à `z` dans la fenêtre d'affichage.

— Réalisez un programme qu' attend des entrées de clavier et qui en fait écho dans la fenêtre d'affichage, mais avec un temps de retard : lorsque vous tapez "Hello", vous verrez ">Hell" et lorsque dans cette séquence vous tapez le "H", ">" apparait, lorsque vous tapez le "e", le "H" apparait et ainsi de suite.

Votre programme bouclera en attendant une entrée en observant l'adresse `0xffff0000`. Une telle attente s'appèlle *busy waiting*.

Vous pouvez ignorer des entrées avec accents et vous limiter à l'encodage ASCII.