# M1 SESI – Architecture des Processeurs et Optimisation

# TD1 – Rappels sur l'architecture du Mips

Le processeur Mips-32 que nous allons étudier au cours de cette UE ne contient que les instructions entières à l'exclusion des instructions de division et de multiplication. L'objectif de ce TD est de vous familiariser avec :

- Le jeu d'instructions du processeur Mips-32
- La syntaxe de l'assembleur
- L'écriture de programme et les conventions utilisées par le compilateur

En principe, ces notions ont été acquises en L2 et L3

## Exercice 1 :

Ecrire une suite d'instructions permettant d'initialiser le registre R5 à 0.

## Exercice 2 :

Ecrire une suite d'instructions permettant de copier le contenu du registre R6 dans R5.

## Exercice 3 :

Ecrire une suite d'instructions permettant d'initialiser le registre R5 à 0x4567 (valeur 16 bits).

## Exercice 4 :

Ecrire une suite d'instructions permettant d'initialiser le registre R5 à 0x4567ABCD (valeur 32 bits).

## Exercice 5 :

strlen est la fonction de la bibliothèque standard du C qui permet de déterminer la longueur d'une chaîne de caractères.

```
int strlen (char * str) {
   int len;
   for (len = 0; str[len] != '\0'; len ++);
   return len;
}
```

Ecrire en assembleur Mips le code de la boucle principale de strlen. On suppose que à l'entrée de la boucle le registre R4 est initialisé avec l'adresse de la chaîne de caractères.

## Exercice 6 :

La fonction strcpy de la bibliothèque standard du C recopie la chaîne de caractères pointée par src à l'emplacement pointé par trg.

```
char * strcpy(char * trg, char * src) {
   int i;
   for (i = 0; src[i] != '\0'; i++) {
      trg[i] = src[i];
   }
   trg[i] = '\0';
   return trg;
}
```

Ecrire en assembleur Mips le code de la boucle principale de strcpy. On suppose que à l'entrée

de la boucle le registre R4 est initialisé avec l'adresse de la chaîne de caractères trg et le registre R5 avec l'adresse de la chaîne de caractères src .

## Exercice 7 :

La fonction strupper remplace dans une chaîne de caractères les caractères minuscules par des caractères majuscules.

```
char * strupper(char * str) {
   int i = 0;
   while (str[i] != '\0') {
      if ((str[i] >= 'a') && (str[i] <= 'z')) {
         str[i] = str[i] - 'a' + 'A'
      }
      i ++;
   }
   return str;
}
```

Ecrire en assembleur Mips le code de la boucle principale de strupper. On suppose que à l'entrée de la boucle le registre R4 est initialisé avec l'adresse de la chaîne de caractères str.

## Exercice 8 :

La fonction addvect réalise la somme pondérée de deux vecteurs d'entiers

```
int * addvect(int * a, int * b, int * c, int size) {
   int i = 0;
   while (size > 0) {
      c[i] = 2 * a[i] + 3 * b[i];
      i++;
      size--;
   }
   return c;
}
```

Ecrire en assembleur Mips le code de la boucle principale de cette fonction. On suppose que à l'entrée de la boucle le registre R4 est initialisé avec l'adresse du tableau a, le registre R5 avec b, le registre R6 avec c et le registre R7 avec size.

### Rappel :

Un certain nombre de conventions sont utilisées par le compilateur (et le système d'exploitation) pour la génération du code assembleur d'un programme. Ici, nous présentons une version inspirée des conventions utilisées par gcc. Une fonction f, au cours de son exécution peut appeler une autre fonction g. Dans ce cas, f est dite la fonction appelante et g la fonction appelée Par ailleurs, une fonction au cours de son exécution peut être amenée à effectuer des calculs. Les résultats de ces calculs sont enregistrés dans les registres. A priori, le processeur Mips dispose de 32 registres pour sauvegarder les résultats. Toutefois, l'utilisation des registres est restreinte par les règles imposées par le compilateur. Le tableau suivant résume les règles d'utilisation des registres.

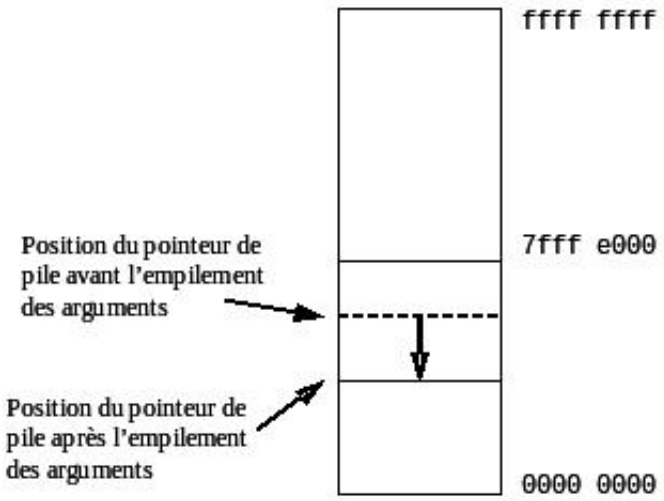
| numéro du registre | Fonction |
|---|---|
| r0 | non modifiable – contient toujours 0 |
| r1 | réservé pour l'assembleur – ne doit pas être utilisé par le compilateur |
| r2 et r3 | valeur de retour de la fonction appelée |
| r4 à r7 | 4 premiers paramètres de la fonction appelée. Si la fonction appelée a plus de 4 paramètres, les autres paramètres sont placés sur la pile |
| | registres de travail non préservés à travers l'appel à une fonction – doivent être |

| | |
|---|---|
| r8 à r15 | sauvegardés par l'appelant |
| r16 à r23 | registres de travail préservés à travers l'appel à une fonction – doivent être sauvegardés par l'appelée |
| r24 et r25 | registres de travail non préservés à travers l'appel à une fonction – doivent être sauvegardés par l'appelant |
| r26 et r27 | réservé pour le système d'exploitation – ne doit pas être utilisé par le compilateur |
| r28 | pointeurs des variables globales |
| r29 | pointeur de pile d'appel des fonctions |
| r30 | registre de travail préservé à travers l'appel à une fonction – doit être sauvegardé par l'appelée |
| r31 | adresse de retour vers la fonction appelante |

Lors de l'appel d'une fonction, la fonction appelante place les arguments de la fonction appelée dans les registres R4 à R7 (dans l'ordre des arguments). Si la fonction appelée a plus de 4 paramètres, les autres arguments sont placés sur la pile.

Au début de la fonction appelante, le pointeur de pile est modifié pour créer suffisamment de places pour les arguments de toutes les fonctions qui seront éventuellement appelées par la fonction. Puis, avant l'appel d'une fonction, les arguments de la fonction appelée sont placés sur la pile. La modification du pointeur de pile consiste à soustraire au pointeur le nombre d'octets nécessaires pour enregistrer les arguments des fonctions appelées (la pile grandit vers les adresses petites). Les arguments sont empilés dans l'ordre inverse de leur apparition dans la liste des arguments de sorte que les premiers arguments se trouvent en haut de la pile.

Les 4 premiers arguments de la fonction appelée ne sont pas placés sur la pile. Ils sont directement disponibles dans les registres R4 à R7 (R4 pour le premier argument et R7 pour le 4ème). Néanmoins, il y a une place réservée pour ces 4 arguments sur la pile. Ainsi, en haut de la pile, après les octets réservés (mais non utilisés) pour les 4 premiers arguments, on trouve dans l'ordre les arguments suivants :




Une fois les arguments empilés, la fonction appelante fait un saut avec sauvegarde de l'adresse de retour (Jal ou une instruction équivalente) à la fonction appelée. Alors le processeur commence à exécuter le code de la fonction appelée.

Les registres de travail sont utilisés par la fonction appelée pour sauvegarder ses calculs intermédiaires. Les règles d'utilisation des registres spécifient deux types de registres de travail : ceux qui doivent être préservés à travers l'appel d'une fonction et ceux qui ne sont pas préservés.

Afin d'éviter que l'exécution de la fonction appelée ne perturbe les valeurs contenues dans les registres de travail à préserver, la fonction appelée commence par sauvegarder sur la pile les registres de travail à préserver qu'il souhaite modifier. Elle commence par déplacer le pointeur de pile, puis empile la valeur des registres. Enfin, les variables locales de la fonction appelante

sont également allouées sur la pile.

A la fin de son exécution, la fonction appelée restitue la valeur des registres de travail à préserver à partir de la pile, restaure le pointeur de pile puis, retourne à la fonction appelante.

## Exercice 9 : Registers R16 onwards

La fonction pgcd calcule le PGCD de deux entiers positifs.

```
int pgcd (int a, int b) {
   while (a != b) {
      if (a < b) {
         b = b - a;
      }
      else {
         a = a - b;
      }
   }
   return a;
}
```

Ecrire en assembleur Mips le code de cette fonction.

## Exercice 10 :

Ecrire en assembleur Mips le code de la fonction tri (tri d'un tableau d'entiers positifs dans l'ordre décroissant – tri max).

```
unsigned int * tri(unsigned int * a, int size) {
   int i;
   int j;
   unsigned int max;
   unsigned int tmp;

   for (i = 0; i < size; i++) {
      max = a[i];
      for (j = i + 1; j < size; j++) {
         if (a[j] > max) {
            tmp = max;
            max = a[j];
            a[j] = tmp;
         }
      }
      a[i] = max;
   }
   return a;
}
```