Corrigé Série de TD n°2

Réponses aux questions de compréhension :

a) Caractéristiques principales du MIPS R3000

- **Architecture 32 bits** : Le R3000 est un processeur 32 bits, ce qui signifie qu'il peut manipuler des mots de 32 bits et possède un bus de données de 32 bits.
- **RISC** : L'architecture repose sur une philosophie RISC, favorisant un jeu d'instructions simple et rapide, avec des instructions de longueur fixe (32 bits) et la plupart des opérations s'exécutant en un cycle d'horloge.
- **Registres** : Le R3000 dispose de 32 registres de travail de 32 bits (de R0 à R31), où R0 est toujours fixé à zéro. Ces registres facilitent le traitement rapide des données.
- **Pipeline** : Il possède un pipeline de cinq étapes : *Fetch*, *Decode*, *Execute*, *Memory Access* et *Write Back*. Ce pipeline permet d'augmenter le débit de traitement en superposant les différentes étapes de l'exécution des instructions.

b) les différents types d'instructions manipulés dans MIPS

Instructions de Type R (Register)

- **Description** : Ces instructions utilisent uniquement des registres pour les opérations, sans faire référence à la mémoire. Elles sont souvent destinées aux opérations arithmétiques, logiques, ou de manipulation de bits.
- **Format** : Les instructions de type R utilisent trois registres (deux opérandes source et un registre destination), un code fonction, et un champ pour spécifier le type d'opération.
- **Exemples** :
    - ADD $t1, $t2, $t3 : Additionne le contenu des registres $t2 et $t3, et stocke le résultat dans $t1.
    - SUB $s1, $s2, $s3 : Soustrait $s3 de $s2 et place le résultat dans $s1.
    - AND $t0, $t1, $t2 : Effectue une opération logique ET entre $t1 et $t2 et stocke le résultat dans $t0.

2. Instructions de Type I (Immediate)

- **Description** : Ces instructions impliquent un registre et une valeur immédiate (un nombre constant). Elles sont souvent utilisées pour les opérations arithmétiques simples et les accès mémoire.
- **Format** : Ce type d'instruction inclut un registre source, un registre destination, et une valeur immédiate de 16 bits.
- **Exemples** :
    - ADDI $t1, $t2, 10 : Ajoute la valeur immédiate 10 au contenu de $t2 et place le résultat dans $t1.
    - LW $t0, 4($s1) : Charge un mot de la mémoire à l'adresse $s1 + 4 dans $t0.
    - SW $t0, 8($s2) : Stocke le contenu de $t0 à l'adresse $s2 + 8.

3. Instructions de Type J (Jump)

- **Description** : Les instructions de type J sont destinées au contrôle de flux, permettant de sauter à une nouvelle adresse. Elles sont utilisées pour les sauts inconditionnels vers des adresses spécifiques.
- **Format** : Elles incluent un champ d'adresse de 26 bits pour spécifier la destination du saut.
- **Exemples** :
    - J 10000 : Sauter à l'adresse 10000.

- JAL 20000 : Jump And Link, saute à l'adresse 20000 et stocke l'adresse de retour dans le registre $ra (registre 31), utile pour les appels de fonctions.

c) Directives de déclaration de données courantes

- **.data** : Indique le début de la section de données. Toutes les données déclarées après cette directive seront stockées dans la section de données.
- **.word** : Réserve de la mémoire pour une ou plusieurs valeurs de 32 bits.
- **.half** : Réserve de la mémoire pour une ou plusieurs valeurs de 16 bits.
- **.byte** : Réserve de la mémoire pour une ou plusieurs valeurs de 8 bits.
- **.asciiz** : Déclare une chaîne de caractères terminée par un caractère nul (\0).
- **.space** : Réserve un espace de mémoire de la taille spécifiée en octets, sans initialiser la valeur.

d) Donnez l'instruction de déclaration pour chaque requête suivante :

- Réservation d'un entier de 32 bits

  var1: *.word 5 # Réserve 4 octets et initialise la variable "var1" à la valeur 5*
- Réservation de plusieurs mots consécutifs initialisés

  tableau*: .word 1, 2, 3, 4 # Réserve 4 entiers (16 octets) et les initialise aux valeurs 1, 2, 3, et 4*
- Réservation d'une chaîne de caractères terminée par un caractère nul

  message*: .asciiz "Hello, MIPS!" # Chaîne terminée par '\0'*
- Réservation de plusieurs octets sans initialisation

  buffer: *.space 16 # Réserve 16 octets de mémoire sans initialiser les valeurs*

Exercice 1 : calcul le périmètre d'un rectangle

```
.data
prompt1: .asciiz "Entrez la longueur: "    # Message pour demander la longueur
prompt2: .asciiz "Entrez la largeur: "     # Message pour demander la largeur
result1: .asciiz "Le périmètre est: "      # Message pour afficher le périmètre
result2: .asciiz "La surface est: "        # Message pour afficher la surface
.text
main:
  # Demander la longueur
  li $v0, 4            # Code pour afficher une chaîne (syscall)
  la $a0, prompt1      # Charger l'adresse de prompt1 dans $a0
  syscall
  # Lire la longueur (entier)
  li $v0, 5            # Code pour lire un entier (syscall)
  syscall
  move $t0, $v0        # Stocker la longueur dans $t0
  # Demander la largeur
  li $v0, 4            # Afficher une chaîne (syscall)
  la $a0, prompt2      # Charger l'adresse de prompt2 dans $a0
  syscall
  # Lire la largeur (entier)
  li $v0, 5            # Lire un entier (syscall)
```

```
syscall
move $t1, $v0          # Stocker la largeur dans $t1
# Calculer le périmètre : P = 2 * (longueur + largeur)
add $t2, $t0, $t1      # $t2 = longueur + largeur
sll $t2, $t2, 1        # $t2 = $t2 * 2 (décalage à gauche de 1 bit)
# Afficher le résultat du périmètre
li $v0, 4              # Afficher une chaîne (syscall)
la $a0, result1        # Charger l'adresse de result1 dans $a0
syscall
move $a0, $t2          # Déplacer le résultat dans $a0 pour affichage
li $v0, 1              # Afficher un entier (syscall)
syscall
# Calculer la surface : S = longueur * largeur
mul $t3, $t0, $t1      # $t3 = longueur * largeur
# Afficher le résultat de la surface
li $v0, 4              # Afficher une chaîne (syscall)
la $a0, result2        # Charger l'adresse de result2 dans $a0
syscall
move $a0, $t3          # Déplacer le résultat dans $a0 pour affichage
li $v0, 1              # Afficher un entier (syscall)
syscall
# Fin du programme
li $v0, 10             # Code pour terminer le programme (syscall)
syscall
```

Exercice n°2 :

Le programme demande d'abord à l'utilisateur de saisir un entier. Ensuite, il affiche le double et le triple de cet entier.

```
.data
prompt: .asciiz "Entrez un entier: "  # Message pour demander l'entier
result1: .asciiz "Le double est: "   # Message pour afficher le double
result2: .asciiz "Le triple est: "   # Message pour afficher le triple
.text
main:
   # Demander l'entier
   li $v0, 4              # Code pour afficher une chaîne (syscall)
   la $a0, prompt         # Charger l'adresse de prompt dans $a0
   syscall
   # Lire l'entier (entier)
   li $v0, 5              # Code pour lire un entier (syscall)
   syscall
   move $t0, $v0          # Stocker l'entier saisi dans $t0
   # Calculer le double : double = 2 * entier
```

```
sll $t1, $t0, 1        # Décale à gauche de 1 bit (équivalent à multiplication par 2)
# Afficher le double
li $v0, 4              # Afficher une chaîne (syscall)
la $a0, result1        # Charger l'adresse de result1 dans $a0
syscall
move $a0, $t1          # Déplacer le double dans $a0 pour affichage
li $v0, 1              # Afficher un entier (syscall)
syscall
# Calculer le triple : triple = 3 * entier
mul $t2, $t0, 3        # Multiplie $t0 par 3 et stocke le résultat dans $t2
# Afficher le triple
li $v0, 4              # Afficher une chaîne (syscall)
la $a0, result2        # Charger l'adresse de result2 dans $a0
syscall
move $a0, $t2          # Déplacer le triple dans $a0 pour affichage
li $v0, 1              # Afficher un entier (syscall)
syscall
# Fin du programme
li $v0, 10             # Code pour terminer le programme (syscall)
syscall
```

Exercice n°3 : Ecrire un programme ***Operations*** en assembleur MIPS permettant de saisir deux entiers et d'afficher successivement leur somme, leur différence, leur produit et leur