Université d'Alger 1

Faculté des Sciences
Département Informatique

Rattrapage L2-ADO

10-Juin-2023

Durée : 90 minutes

Nom : Prénom : Groupe : Matricule :		/ 20

Exercice 1 : (4 points) (- 0.25 point pour chaque réponse fausse)

Mettez ce qui suit dans l'ordre chronologique (1,2,3,…). Nous l'avons commencé pour vous.

6	Le Code et les Données de différents endroits sont regroupés en un seul fichier.
1	Un étudiant L2-INFO se voit attribuer une tache pour mettre en œuvre un projet `'big_nums'`.
9	L'exécution commence à : `main`.
2	L'étudiant-e écrit son code en C.
5	Les tables des symboles sont produites.
4	Les pseudo-instructions assembleur sont traduites en instructions ISA.
8	Les espaces pour **données statiques**, **globale** et le **code** sont réservés / initialisés en mémoire.
3	Le code C de l'étudiant-e est traduit en MIPS.
7	L'édition des liens est effectuée.

Exercice 2 : (4 points)

A/ La pseudo instruction assembleur **nop** peut être traduite en véritable instruction de l'ISA du MIPS de différentes manières. Donnez **deux traductions possibles** qui ont des opcodes ou des champs de fonction différents.

(1 point)

```
slt $0, $0, $0    ou    add $0, $0, $0    ou encore    addi $0, $0, 0
```

B/ Un concepteur logiciel aimerait ajouter une nouvelle pseudo instruction au langage assembleur du MIPS. L'instruction, appelée **`lwa`** (pour *load word from address*), prend la forme :

```
lwa $rd, addr
```

Le premier argument est un registre de destination. Le deuxième argument est une constante de 32 bits (tout comme le deuxième argument de la pseudo instruction **`la`**). L'instruction doit charger le mot 32 bits depuis l'emplacement mémoire indiqué par **`addr`** et mettre cette valeur dans le registre **`$rd`**.

Montrez comment l'assembleur doit traduire cette pseudo-instruction en vrai(es) instruction(s) de l'ISA MIPS. (C.-à-d. Votre code doit contenir les paramètres **`$rd`** et **`addr`** quelque part et doit être constitué entièrement d'instruction(s) véritable(s) du MIPS).

(3 points)

```
lui $at, addr[31:16]                lui $rd, addr[31:16]
ori $at, $at, addr[15:0]     ou     ori $rd, $rd, addr[15:0]
lw  $rd, 0($at)                     lw  $rd, 0($rd)
```

Exercice 3 : (4 points) (- 0.25 point pour chaque réponse fausse)

Soit un cache direct de 4 blocs (lignes de cache) au total où chaque bloc est constitué de 1 octet. Nous aimerions utiliser ce cache dans un ordinateur avec des adresses 8 bits et un adressage d'octets – c.-à-d. chaque adresse 8 bits référence un seul octet dans la mémoire. Le cache est initialement vide.

Le processeur exécute une séquence d'instructions de chargement à partir d'une liste d'adresses mémoire dans l'ordre gauche-droite indiqué dans le tableau ci-dessous. Pour chacune de ces instructions, indiquez si le résultat est un succès

ou un échec, en écrivant **succès** ou **échec** dans la boîte. Aussi, pour chaque échec, indiquez la raison comme étant **conflictuelle** ou **obligatoire**.

Adresse (en hexa)	0x00	0x02	0x01	0x03	0x04	0x02	0x01	0x0B
Succès / Echec	échec	échec	échec	échec	échec	succès	succès	échec
Raison	obligatoire	obligatoire	obligatoire	obligatoire	conflit			conflit

Exercice 4 : (8 points)

Ce qui suit est une implémentation inefficace d'une fonction **fun** dans le langage assembleur MIPS. Lisez le code attentivement et répondez aux questions ci-dessous.

```
fun:
        mov     $v0, $0
        li      $s0, 1
loop:   beq     $a1, $0, end
        addiu   $a1, $a1, -1
        sll     $s0, $s0, 1
        div     $a0, $s0
        mfhi    $s1
        or      $v0, $v0, $s1
        j       loop
end:    jr      $ra
```

(2 points)

```
// supposez que : y < 31

unsigned int fun(unsigned int x, unsigned int y) {


   return x & ( (1 << y) - 1 );



}
```

A/ Brièvement, expliquez ce que **fun** retourne (en supposant que y < 31). Ne décrivez pas l'algorithme, expliquez seulement comment la valeur retournée se rapporte aux paramètres en entrée x et y.

(2 points)

Retourne les 'y' bits inférieurs de x.

B/ Écrivez du code C optimisé pour la fonction **fun** dans la boîte (rendez-le aussi compact et efficace que possible pour une note complète).

C/ Euh oh ! Nous avons enfreint certaines conventions d'appel dans le code ci-dessus ! Que devrions-nous ajouter au début (avant l'instruction **mov**) et à la fin (avant l'instruction **jr**) de la fonction **fun** pour corriger cela ?

Début **(2 points)**	**Fin** **(2 points)**

`addiu  $sp, $sp, -8`
`sw $s0, 0($sp)`
`sw $s1, 4($sp)`
	`lw $s0, 0($sp)`
	`lw $s1, 4($sp)`
	`addiu  $sp, $sp, 8`