Université d'Alger 1

Faculté des Sciences
Département Mathématiques et Informatique

Rattrapage L2-ADO
Decembre-2020

Nom : Prénom : Matricule :		/ 20

Exercice 1 : (9 points)

Considérer les définitions de données suivantes :

```
        .data
var1:       .byte       'd'
var2:       .word       0xdad, 0xdeadbeef, 128
var3:       .byte       -1, 0, 'z'
var4:       .half       256, 1, 0xffff
str1:       .asciiz     "L2AdO"
```

1/ Donner le contenu de chaque octet de la mémoire allouée **en hexadécimal** pour les définitions de données ci-dessus. L'ordre **Little Endian** est utilisé pour les octets des mots et demi-mots (half). (0.25 point par case correcte)

2/ Remplir la table des symboles en indiquant **toutes les étiquettes** et leurs **adresses de début**. (1 point au total)

3/ Indiquez par un **X** dans la case les octets ignorés (mais alloués) dans le segment de données. (0.25 point par case correcte)

Segment de données

Adresse	Octet 0	Octet 1	Octet 2	Octet 3
0x10010000	0x64	X	X	X
0x10010004	0xad	0x0d	0x00	0x00
0x10010008	0xef	0xbe	0xad	0xde
0x1001000C	0x80	0x00	0x00	0x00
0x10010010	0xff	0x00	0x7a	X
0x10010014	0x00	0x01	0x01	0x00
0x10010018	0xff	0xff	0x4c	0x32
0x1001001C	0x41	0x64	0x4f	0x00
0x10010020				
0x10010024				
0x10010028				
0x1001002C				

Table des symboles

Etiquette	Adresse
var1	0x10010000
var2	0x10010004
var3	0x10010010
var4	0x10010014
str1	0x1001001A

4/ Combien d'octets sont alloués au total (incluant les octets ignorés) dans le segment de données ? (0.25 point)

32 octets

Exercice 2 : (4 points)

Dans une équipe de développement logiciel, votre patron vous a attribué la tache d'assembler un bout de code en assembleur MIPS selon les spécifications de l'ISA, et ceci afin de déboguer la sortie d'un compilateur en développement par une équipe alternative.

Complétez le codage des instructions assembleurs indiquées à droite dans le tableau (4 points)

Adresse de l'instruction	Code machine	Code assembleur
		`# Instructions assembleur à coder.`
0x0040 0000	0x00048021	`addu $s0 , $zero , $a0`
0x0040 0004	0x00058821	`addu $s1 , $zero , $a1`
0x0040 0008	0x24020000	`addiu $v0 , $zero , 0`
0x0040 000C	0x24030000	`addiu $v1 , $zero , 0`
0x0040 0010	**0x1200000a** <-- (0.5 pt)	`L1: beq $zero , $s0 , FIN`
0x0040 0014	**0x8e280000** <-- (0.5 pt)	`lw $t0 , 0($s1)`
0x0040 0018	**0x31180001** <-- (0.5 pt)	`andi $t8 , $t0 , 1`
0x0040 001C	**0x17000002** <-- (0.5 pt)	`bne $zero , $t8 , L2`
0x0040 0020	0x20420001	`addi $v0 , $v0 , 1`
0x0040 0024	**0x0810000b** <-- (0.5 pt)	`j L3`
0x0040 0028	0x20630001	`L2: addi $v1 , $v1 , 1`
0x0040 002C	**0x22310004** <-- (0.5 pt)	`L3: addi $s1 , $s1 , 4`
0x0040 0030	0x20010001	`addi $at , $zero , 1`
0x0040 0034	0x02018022	`sub $s0 , $s0 , $at`
0x0040 0038	**0x08100004** <-- (0.5 pt)	`j L1`
0x0040 003C	**0x03e00008** <-- (0.5 pt)	`FIN: jr $ra`

Exercice 3 : (7 points)

1/ Le pipelining améliore-t-il la latence des instructions individuelles ? Justifiez ! (1 point)

Le pipelining n'améliore PAS la latence des instructions individuelles. Mais, il améliore le débit.

2/ Donnez les causes des aléas de contrôle dans un chemin de données en pipeline et comment ils sont gérés ? (2 points)

Les aléas de contrôle sont causés par des instructions de saut et de branchement retardées dans un chemin de données en pipeline. Ils peuvent être éliminés en convertissant les prochaines (une ou deux) instructions qui apparaissent après un saut ou une branche prise en instruction NOP.

3/ Expliquer les concepts de localité temporelle et de localité spatiale dans la mémoire cache. (2 points)

Localité temporelle : si un programme fait référence à une instruction (ou à des données) à une adresse spécifique, il pourrait à nouveau faire référence à cette même adresse dans l'avenir.

Localité spatiale : si un programme fait référence une instruction (ou à des données) à une adresse spécifique, il y a de fortes chances qu'il référence la prochaine adresse en mémoire à l'avenir.

4/ Expliquez la différence entre un cache « write-through » et un cache « write-back ». (2 points)

Cache « write-through » : Chaque écriture dans le cache est écrite dans la mémoire de niveau inférieur.

Cache « write-back » : l'écriture se fait uniquement dans le cache. Un bit « dirty » est nécessaire pour indiquer si un bloc a été modifié. Les blocs modifiés sont réécrits en mémoire lorsqu'ils sont remplacés.