TP1 et 2 ASR1 **Codage et Introduction processeur MIPS(ASR2)**

A.Initiation

Code 1.

Algorithme

```
entier x,a=30 ;
Début ; x←a  ; Fin
```

Code Source .asm (MIPS)

```
        .data 0x10000000
a :     .word 30
x :     .space 4
        .text
        .globl __start

__start:
lw $2,a
sw $2,x
```

Code 2

Algorithme

```
entier a=20,x=10,c ;
Début ; c←a-x ; Fin
```

Code Source .asm (MIPS)

```
        .data 0x10000000
a :     .word 20
x :     .word 10
c :     .space 4

        .text
        .globl __start

__start:
lw $2,a
lw $3,x
sub $4,$2,$3
sw $4,c
```

Code 3

Code Source .asm (MIPS)

```
        .data 0x10000000
stock:  .word 12,12,2,4,7

        .text
.globl __start

__start:
        li $19,0
Loop:   lw $8,stock($19)    #$8<---Memoire[stock+$19]
        bne $8,12,Exit
        addi $19,$19,4
        j Loop
Exit:
```

Ce programme gère, un indice (variable i) qui correspond au registre $19 et un tableau stock de 5 mots qui débute à l'adresse 0x10000000.

Stock :

Valeur	12	12	2	4	7
Adresse	0x10000000	0x10000004	0x10000008	0x1000000C	0x10000010

Les ressources de données gérées par le processeur et son langage sont :

Mémoire (utilisateur)

type	adresse	valeur
text	0x4000000	32bits
	0x40.....4	….
	…..	….
data	0x10000000	…..
	0x10000004	
	0x10000008	
pile	0x7FFEFFC	

Registres

numéro	valeur	nom	Convention d'usage
$0	32bits		Réservé
$1	32bits		Réservé
$2	….	Cf doc	Cf doc
$3		...	...
...			
$31			

Exercices

Pour chacun des 3 codes ,

1.Donnez le code brut asm (celui qui apparaît sur le simulateur en tant que code réellement exécuté)

2.Donnez le code exécutable

3 Déroulez le programme pas à pas et renseignez le tableau suivant en donnant, pour chaque ligne exécutée, les changements générés en mémoire et/ou sur les registres .

4 Expliquez chaque ligne du code .asm

Pour le code n°3 ,

3 Déroulez tous les passages de la boucle

5 Corrigez l'algorithme suivant pour qu'il corresponde au code 3 : Stock tableau de 5 entiers, entier i Début ; i←1 ; Tant que (stock[i]!=12) faire i←i+1 FinTantQue ; Fin

Adresse instruction et/ou label	Résultat Ligne	Mémoire			Registre		Référence de saut (label et adresse)
		adresse	valeur	nom	numéro	valeur	
	Initialisation (.data ou set values)						
	a: .word 30	0x10000000	1E	a			
	X :.space 4	0x10000004		x			
	Code (.text)						
0x400000	lw $2,a				$2	1E	
0x400004	Sw $2,x	0x10000004	1E	x			

B.codage

0.
Introduction : Lire le paragraphe 2.3 page 12 de la documentation MIPS R2000 Simulator. Quel était le mode d'adressage utilisé dans le code 2 ?, dans le code 3 ? .

1.
Réalisez l'addition de [0x10000000] et de [0x10000004] et stockez le résultat en [0x10000008] en utilisant pour lw et sw le mode d'adressage dit immédiat qui consiste à donner les adresses explicitement dans l'instruction.
Ici faites un change memory content sur le simulateur pour initialiser les valeurs de [0x10000000] et de [0x10000004].

2.
Réalisez le même programme mais avec l'adressage symbolique utilisé dans les codes 1 et 2 de la partie A.

3.
On veut $a0 ← octet [0x10000000] en utilisant les instructions lb et lbu.

Choisir un entier signé positif et un entier signé négatif (raisonner sur 8 bits en hexadécimal).
Inscrivez chacune des ces 2 valeurs en [0x10000000] (click droit et Change Memory Content) puis relevez le résultat obtenu dans $a0 pour lb et lbu. Ce qui revient à complétez le tableau suivant :

			$a0 mot = 4 octets	
			pour lb	pour lbu
octet [0x10000000]	Signé positif	?	?,?	?,?
	Signé négatif	?	?,?	?,?

N.b. : ?,?= valeur sur 32 bits (4 octets) , signe

Conclure : lb et lbu conservent-elles le signe de l'octet chargé ?

4.
Même question que 1 mais l'on veut additionner 2 octets seulement et ainsi utiliser moins d'espace mémoire, soit faire [0x10000002]← [0x10000001] + [0x10000000].
L'initialisation peut se faire par un Change Memory Content (adresse=0x10000000, value=0x00000708) pour faire 7+8.
Indiquez combien d'octets mémoires ont été utilisés ici et comparez aux nombres d'octets utilisés en 1.
En regard, indiquez par le calcul la plus grande valeur possible pour le résultat ici et en 1.

5.
Gestion de la retenue dans l'addition $a3 ← $a2+$a1.
Bien que l'on ait 32 bits de données, la plus grande valeur chargeable dans un registre est 7FFFFFFF.
Le 32^{ième} bit est le bit de retenue dans l'addition.

Que se passe-t-il si l'addition est effectuée par l'instruction add, si elle est effectuée par addu ?
Le calcul a t-il lieu dans tous les cas ?
Répondez à ces questions en choisissant un jeu de valeur pour $a1 et $a2 (que vous chargez directement par li) et en relevant le résultat obtenu dans $a3 pour add et addu. Pour ce faire, complétez le tableau suivant :

	$a1	$a2	$a3	
			pour add	Pour addu
somme<=7FFFFFFF	?	?	?	?
somme>7FFFFFFF	?	?	?	?

6.
Que font les instructions div et rem.
Les utiliser pour donner le résultat de la division euclidienne de 223/12.
Donnez le code que vous avez utilisé ainsi que la valeur des registres qui contiennent votre résultat.

7.
Que font les instructions rol ror srl and et neg ?
A partir de la documentation, expliquez-les.
Ecrivez un code qui les mettent en œuvre avec à chaque fois (0xA000000C, 4) (*ou seulement 0xA000000C pour neg) comme valeurs de test. Relevez les résultats obtenus dans les registres destinations.

	rol dans $3	ror dans $4	srl dans $5	and dans $6	neg dans $7
0xA000000C, 4*					

Vérifiez chaque résultat obtenu par le calcul

8.
Utilisez le coprocesseur flottant pour donner les représentations de 1.5. Vérifiez le résultat. La norme de codage utilisée est la IEEE 754 vue en TD.

li.s $f3,1.5 3FC0 0000 == 0011 1111 1100 000000000000...00000
s exposant mantisse

9.
Commentez les 2 lignes qui traduisent la pseudo instruction li.s … en instructions brutes. Qui est responsable du codage en flottant, l'assembleur ou le coprocesseur ?

10.
Additionnez 2.5 avec 1.3e2 (130). Donnez le codage des opérandes et du résultat. Montrez que le codage du résultat est juste.

11.
Reprise de la question 5. Dans le cas où l'addition provoque une exception, donnez les valeurs des registres du coprocesseur0 Status Cause et EPC. A quel code et type d'exception correspond Cause (CM,$3) et annexe 3.

12.
Modifiez Status pour masquer les 8 niveaux d'interruption. Utiliser l'instruction mtc0 qui transfert un registre UC vers le coprocesseur 0. Les informations nécessaires ont été données en 0.

13.
Ecrire un programme qui utilise div et rem (voir Question 6) pour donner avec une précision de 3 décimales le résultat de n1/n2 avec n1,n2 < 10.