TD7 : – Architecture logicielle :
Instructions mémoire et instructions arithmétiques

Dans tout le TD, on considère le jeu d'instructions MIPS32. Le jeu d'instructions est défini en annexe. Les explications sur le jeu d'instructions seront fournies au fur et à mesure.
Dans ce TD, on considérera essentiellement les instructions mémoire et les instructions arithmétiques.

1. Format des instructions

Donner le codage hexadécimal des instructions suivantes

a) ADD $r3,$r1,$r2
b) ADDI $r2,$r1, -1
c) SLL $r2,$r1,4
d) SRL $r2,$r1,12
e) SRA $r2,$r1,8
f) LUI $r2, 0xABCD

Quel est l'intervalle des entiers représentables par le champ immédiat d'une instruction en format I ?

2. Etude des instructions

Initialement, le registre r1 contient 0x95842103 et le registre r2 contient 0x00001000
En repartant à chaque fois de l'état initial, donner l'état des registres modifiés après exécution des instructions

a) ADD $r3,$r1,$r2
b) ADDI $r3, $r1, -1
c) SLL $r2,$r1,4
d) SRL $r2,$r1,8
e) SRA $r2,$r1,12
f) LUI $r20xABCD
g) ORI $r2,$r1,0xAB CD

3. Suite d'instructions

- Quel est le résultat des deux séquences d'instructions suivantes :

  Séquence 1

  ```
  LUI $r1, 0x9876
  ADDI $r1,$r1, 0x8432
  ```

  Séquence 2

  ```
  LUI $r1, 0x9876
  ORI $r1,$r1, 0x8432
  ```

- Ecrire une séquence d'instructions qui positionne le registre r2 à 0x9876
- Ecrire une séquence d'instructions qui positionne le registre r2 à 0x12348765
- Ecrire un programme qui multiplie par 65 le contenu du registre r1 interprété en non signé, avec résultat dans r2 (sans considérer le problème du dépassement).
- *(optionnel) Ecrire un programme qui multiplie par 35 le contenu du registre r1 interprété en non signé, avec résultat dans r2 (sans considérer le problème du dépassement).*

4. Programmation

Exercice 1

Donner la suite d'instructions pour exécuter le code C suivant, où les variables A, B et C sont en mémoire à partir de l'adresse 0x1000 0000.

```
int A, B, C ;
A=B+C ;
```

Exercice 2

Donner la suite d'instructions pour exécuter le code C suivant, où les variables D, E et F sont en mémoire à partir de l'adresse 0x1000 0010.

```
char D,E,F ;
D=E+F ;
```

Exercice 3

Donner la suite d'instructions pour exécuter le code C suivant, où les variables H, J, K sont en mémoire à partir de l'adresse 0x1000 0020.

```
unsigned char H,J,K ;
H=J+K ;
```

Exercice 4

Donner la suite d'instructions pour exécuter le code C suivant, où les variables L, M, N sont en mémoire à partir de l'adresse 0x1000 0030.

```
short L,M,N ;
L=M+N;
```

5. Instructions utilisées

Formats

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Les instructions avec code opération et fonctions.

Name	Instruction syntax	Meaning	Format/opcode/funct			Notes/Encoding
Add	add $d,$s,$t	$d = $s + $t	R	0	20_{16}	adds two registers, executes a trap on overflow 000000ss ssssssss ddddd--- --100000
Add immediate	addi $t,$s,C	$t = $s + C (signed)	I	8_{16}	-	add sign-extended, executes a trap on overflow 001000ss ssssssss CCCCCCCC CCCCCCCC

Load word	lw $t,C($s)	$t = Memory[$s + C]	I	23_{16}	-	loads the word stored from: MEM[$s+C] and the following 3 bytes.
Load halfword	lh $t,C($s)	$t = Memory[$s + C] (signed)	I	21_{16}	-	loads the halfword stored from: MEM[$s+C] and the following byte. Sign is extended to width of register.
Load halfword unsigned	lhu $t,C($s)	$t = Memory[$s + C] (unsigned)	I	25_{16}	-	As above without sign extension.
Load byte	lb $t,C($s)	$t = Memory[$s + C] (signed)	I	20_{16}	-	loads the byte stored from: MEM[$s+C].
Load byte unsigned	lbu $t,C($s)	$t = Memory[$s + C] (unsigned)	I	24_{16}	-	As above without sign extension.
Store word	sw $t,C($s)	Memory[$s + C] = $t	I	$2B_{16}$	-	stores a word into: MEM[$s+C] and the following 3 bytes. The order of the operands is a large source of confusion.
Store half	sh $t,C($s)	Memory[$s + C] = $t	I	29_{16}	-	stores the least-significant 16-bit of a register (a halfword) into: MEM[$s+C].
Store byte	sb $t,C($s)	Memory[$s + C] = $t	I	28_{16}	-	stores the least-significant 8-bit of a register (a byte) into: MEM[$s+C].

Load upper immediate	lui $t,C	$t = C << 16	I	F_{16}	-	loads a 16-bit immediate operand into the upper 16-bits of the register specified. Maximum value of constant is 2^{16}-1
Or immediate	ori $t,$s,C	$t = $s | C	I	D_{16}	-	Leftmost 16 bits are padded with 0's

Shift left logical	sll $d,$t,shamt	$d = $t << shamt	R	0	0	shifts shamt number of bits to the left (multiplies by 2^{shamt})
Shift right logical	srl $d,$t,shamt	$d = $t >> shamt	R	0	2_{16}	shifts shamt number of bits to the right - zeros are shifted in
Shift right arithmetic	sra $d,$t,shamt	$d = $t >> shamt	R	0	3_{16}	shifts shamt number of bits - the sign bit is shifted in