

1[mode d'adressage] Pour chaque instruction, indiquez quel mode d'adressage est utilisé pour la seconde opérande.

- a) ADD R0, #4, R2:
- b) LOAD R0, 8000:
- c) ADD R0, R1, R2:
- d) LOAD R0, 8(R1):
- e) LOAD R0, (R1) :

2. [Modèle d'exécution] Indiquer un modèle d'exécution pouvant correspondre à chacune des instructions suivantes (donner un modèle par instruction)


- a) ADD R1,R2:
- b) ADD (R1),(R2),(R3):
- c) ADD R1, R2, R3:
- d) ADD (R1),R2:

3. [Modèle Exécution] Le processeur RISC-V est un processeur bâti sur le modèle (3,0), répondez par vrai, faux ou ne sais pas (une réponse fausse implique un malus) aux affirmations suivantes concernant le processeur RISC-V.


- a) Seules les instructions load et store peuvent accéder à la mémoire :
- b) L'instruction addition peut avoir 3 registres comme opérandes :
- c) L'instruction addition du processeur RISC-V peut avoir une opérande se trouvant directement en mémoire :

4. Jeu d'Instruction RISC-V 5,5 pts

Le programme assembleur ci-dessous provient d'un processeur RISC-V (ici l'architecture 64 bits). Les instructions utilisées sont décrites à la suite du code. Les registres sont notés X, le registre X0 est un registre qui contient toujours la valeur 0.

a.  Le processeur RISC-V est un processeur bâti sur le modèle (3,0). Qu'est-ce que ça implique ? (1)

b. Quels sont les différents types d'adressage utilisés dans ce code ? Illustrer votre réponse. (1)

c. Quel est le lien entre l'instruction *b/t* et le registre d'état ? (1) 

d. Traduire en langage C le programme en assembleur ci-dessous. **Commenter votre code pour faire explicitement le lien avec le code assembleur.** (2,5) Votre code devra être simple et compact.

```
addi x6,x0, 0
```

```
addi x5,x0, 0
```

```
addi x29, x0, 100
```

```
LOOP : ld x7, 0(x10)
```

```
add x5,x5, x7
```

```
addi x10,x10,8
```

```
addi x6,x6,1
```

```
b/t x6, X 29, LOOP
```

```
sd x5,0(x11)
```

Instruction	Nom	Description
BLT rs1,rs2,Label	Branch Less Than	if rs1 < rs2 then pc ← Label
ADDI rd,rs1,imm	Add Immediate	rd ← rs1 + imm
ADD rd,rs1,rs2	Add	rd ← rs1 + rs2
LD rd,offset(rs1)	Load Double	rd ← Mem[rs1 + offset]
SD rs2,offset(rs1)	Store Double	Mem[rs1 + offset] ← rs2