# 1.)

J’assume une cache avec des adresses de 32-bits, conçue comme à la Figure 1, avec un temps d’accès de 10 cycles d’horloges. Aucun test en écriture ne sera réalisé.

|  |  |  |
| --- | --- | --- |
| But du test | Opération | Résultat attendu |
| Valider le fonctionnement de la cache | Lire la donnée à l’adresse 0x10010000  Lire à nouveau la donnée à l’adresse 0x00000000 | La 1e lecture prend 10 cycles d’horloge, mais la 2e prend 1 cycle. |
| Valider que toutes les données d’un bloc sont bien dans la cache | Lire la donnée à l’adresse 0x10010004 | La lecture ne devrait prendre qu’un seul cycle d’horloge. |
| Valider l’adressage au niveau d’un seul bloc | Lire la donnée à l’adresse 0x10010002 | La lecture ne devrait prendre qu’un seul cycle d’horloge, et retourner la bonne donnée pour le 3e octet dans le bloc. |
| Remplir la cache pour la suite des tests | Lire les données aux adresses suivantes :   * 0x10010010 * 0x10010020 * 0x10010030 * 0x10010040 * 0x10010050 * 0x10010060 * 0x10010070 | Chaque lecture devrait prendre 10 cycles d’horloges à faire.  Cette opération n’est pas un test en soi mais une étape dans la réalisation des tests. |
| Valider que les données initiales sont encore présentes | Lire la donnée à l’adresse 0x10010000 | La même donnée qu’à la première étape, avec un délai d’un seul cycle d’horloge. |
| Valider le remplacement des données | Lire la donnée à l’adresse 0x10010080 à 2 reprises. | La première lecture devrait prendre 10 cycles d’horloge, et la 2e lecture de n’en prendre qu’une. |
| Valider que la bonne donnée a été remplacée | Lire la donnée à l’adresse 0x10010010 | La lecture devrait prendre 10 cycles d’horloge. |

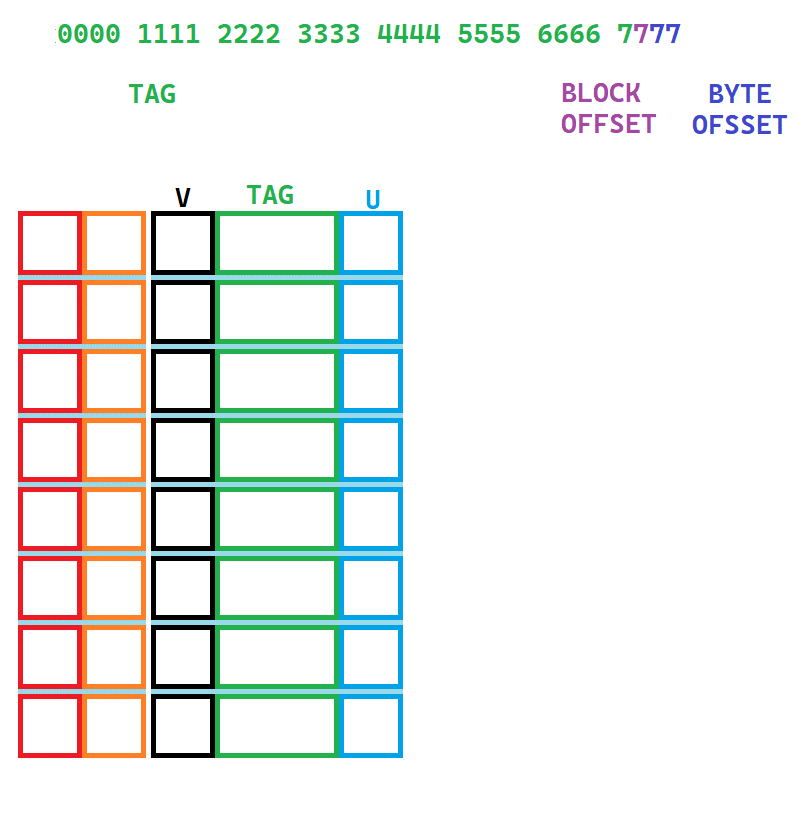


Figure - Cache Q1

# 2)

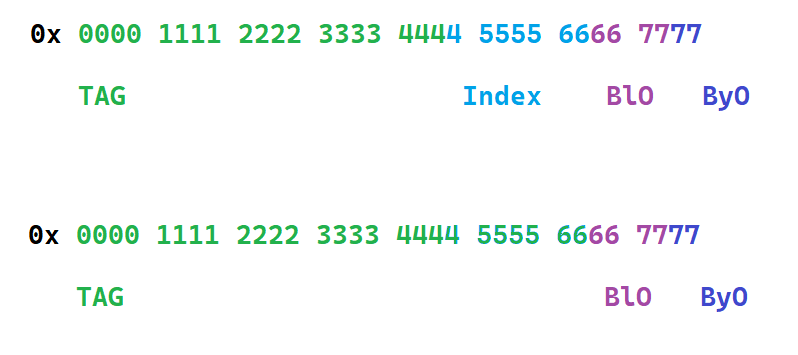


Figure - Cache Q2

# 3)

On cherche à avoir le chiffre de performance le plus faible, et le est fixe à 35%.

Table

Description automatically generated

Figure - Calculs performance Q3

La taille de bloc optimale serait de 16.

# 4)

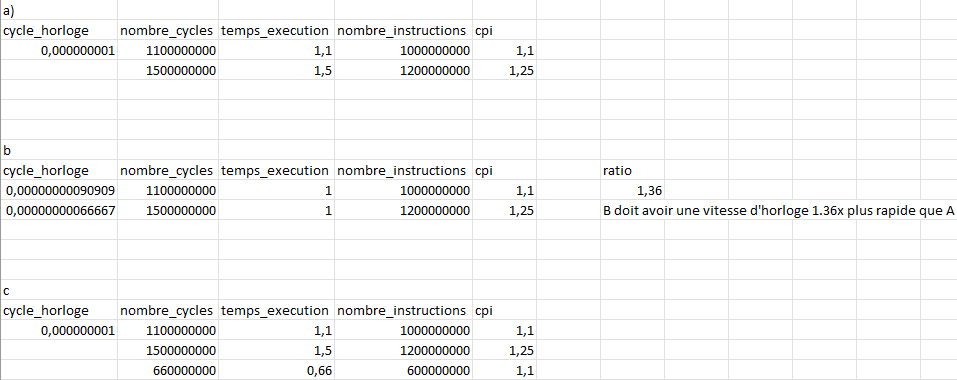


Figure - Calculs performance compilateurs

# 5)

a)

Il s’agit d’une instruction lw ou sw accédant directement à un label ou une adresse comme valeur immédiate, ce qui ne fonctionne pas car les adresses sont 32 bits mais les immediates font 16 bits. (un lw avec un mode d’adressage non-supporté prend donc 3 instructions)

b)

li $t0 0

lui $t0 1001

lw $t0 0($t0)

sw $t0 248($t0)

c)

le sra est est décalage arithmétique qui garde le signe intact

# 6)

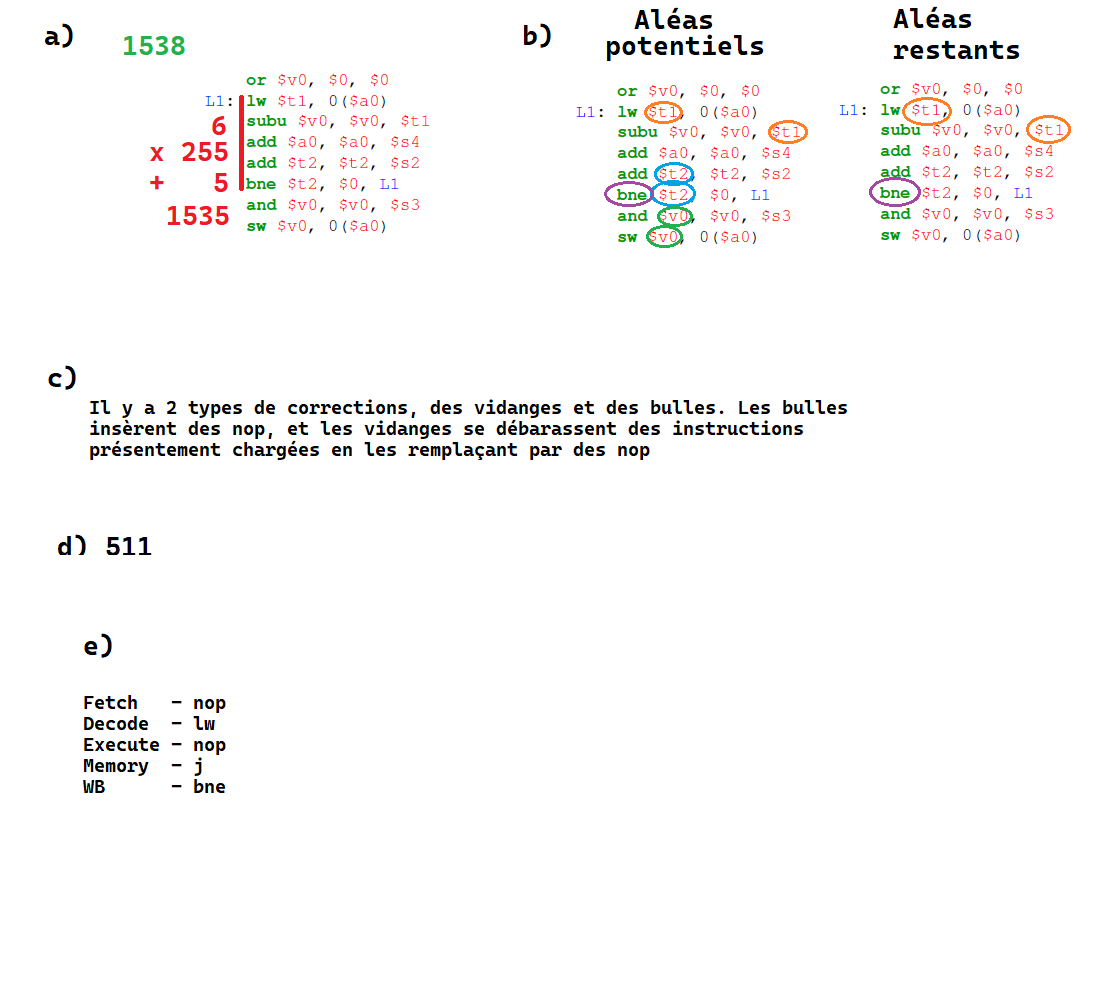


Figure - Question 6

# 7)

a) l’apocalypse

b)

lw $t0, 40($6)

nop

nop

add $t1, $t3, $t0

add $t0, $t5, $t4

nop

sw $t1, 20($t4)

and $t0, $t0, $t4

c)

bulle <= ‘1’ when RegDst = ‘1’ and (ID\_EX.WriteRegister = IF\_ID.WriteRegistrer or EX\_MEM.WriteRegister = IF\_ID.WriteRegister) else ‘0’;

# 8)

Ajouter un mux à l’entrée du program counter, qui est contrôlée par un nouveau signal de contrôle « JumpRegister ». L’entrée A du mux est l’entrée normale du PC, et l’entrée B du mux est la donnée du registre provenant de ReadData1.

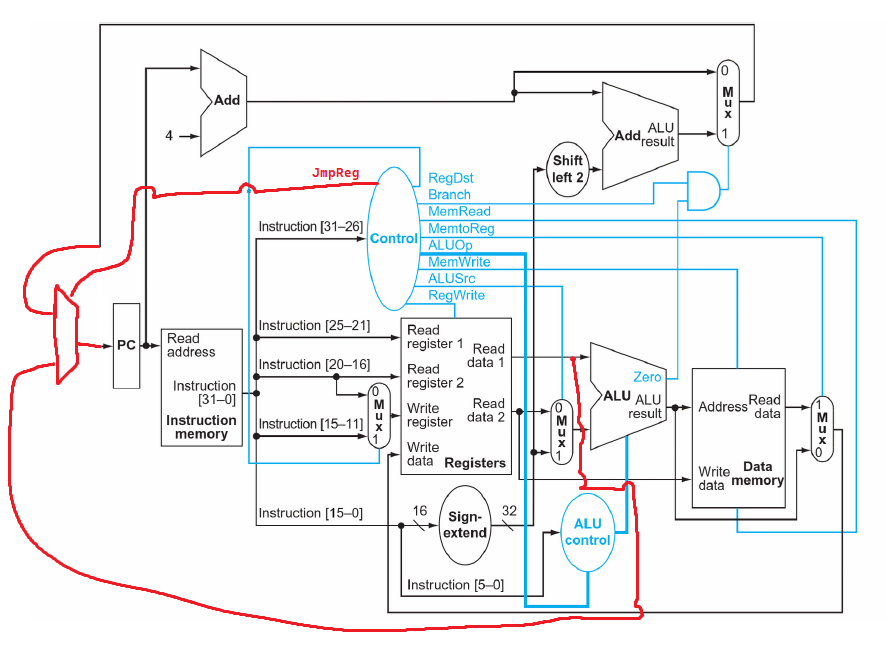


Figure - Modifications apportées au processeur pour rajouter l'instruction jr

|  |  |
| --- | --- |
| Signal de contrôle | État |
| ALUOp | NULL |
| JmpReg | 1 |
| RegDst | 0 |
| MemRead | 0 |
| MemToReg | 0 |
| MemWrite | 0 |
| ALUSrc | 0 |
| RegWrite | 0 |