Question 1

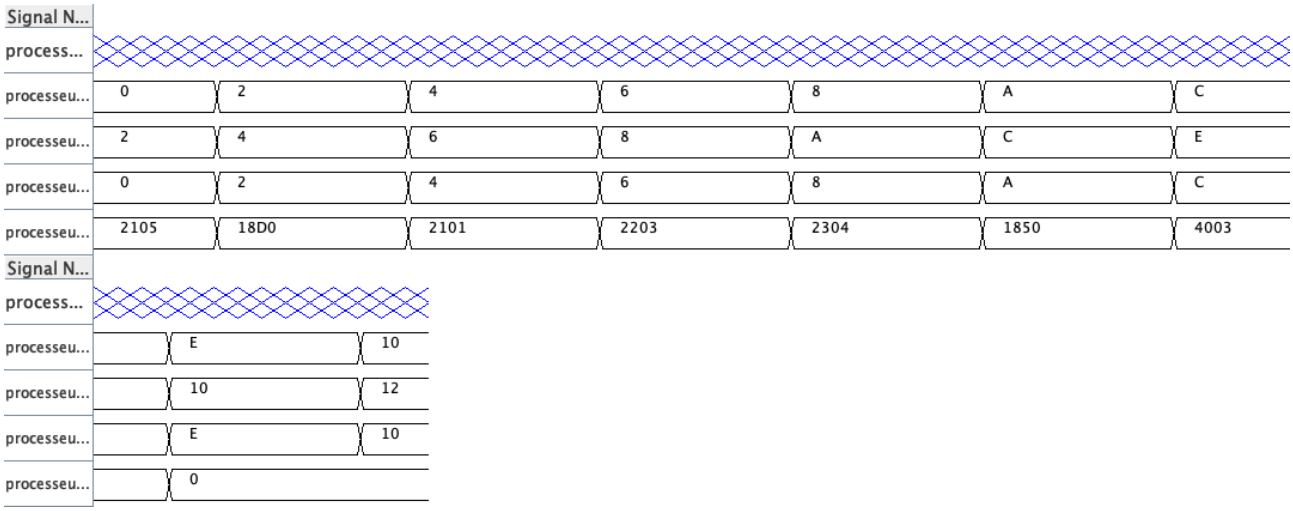
La valeur de l'incrément est de **2** étant donné qu'on veuille réaliser une architecture 16 bits.

Question 2

Si l'architecture est sur 32 bits, la valeur de l'incrément du PC serait de **4** parce Dans un processeur 32 bits, chaque emplacement mémoire est généralement adressé par des mots de **32 bits** (soit 4 octets). Cela signifie qu'un pointeur ou une adresse mémoire est souvent incrémenté de **4** pour accéder à la prochaine valeur de 32 bits en mémoire.

Question 3

1- Les deux signaux identiques sont **`PC_pres_i`** et **`inst_addr_o`** parce que Le processeur



utilise **`PC_pres_i`** pour accéder à l'adresse mémoire et récupérer l'instruction via **`inst_addr_o`**, qui est initialement égal à **`PC_pres_i`**. Après la récupération de l'instruction, **`PC_pres_i`** est mis à jour (généralement incrémenté de 4 ou 2 selon l'architecture). Enfin, **`PC_fut_o`** contient l'adresse de l'instruction suivante, calculée à partir de **`PC_pres_i`**.

2-

`mov r1, #5` : Cette instruction charge la valeur 5 dans le registre `r1`. Elle peut être représentée par un code binaire comme `0xE3A01005`.

`add r0, r2, r3` : Cette instruction additionne les valeurs de `r2` et `r3` et place le résultat dans `r0`. Son code binaire pourrait être `0xE0800003`.

`mov r1, #1` : Cette instruction charge la valeur 1 dans le registre `r1`. Son code binaire pourrait être `0xE3A01001`.

`mov r2, #3` : Cette instruction charge la valeur 3 dans le registre `r2`. Son code binaire pourrait être `0xE3A02003`.

`mov r3, #4` : Cette instruction charge la valeur 4 dans le registre `r3`. Son code binaire pourrait être `0xE3A03004`.

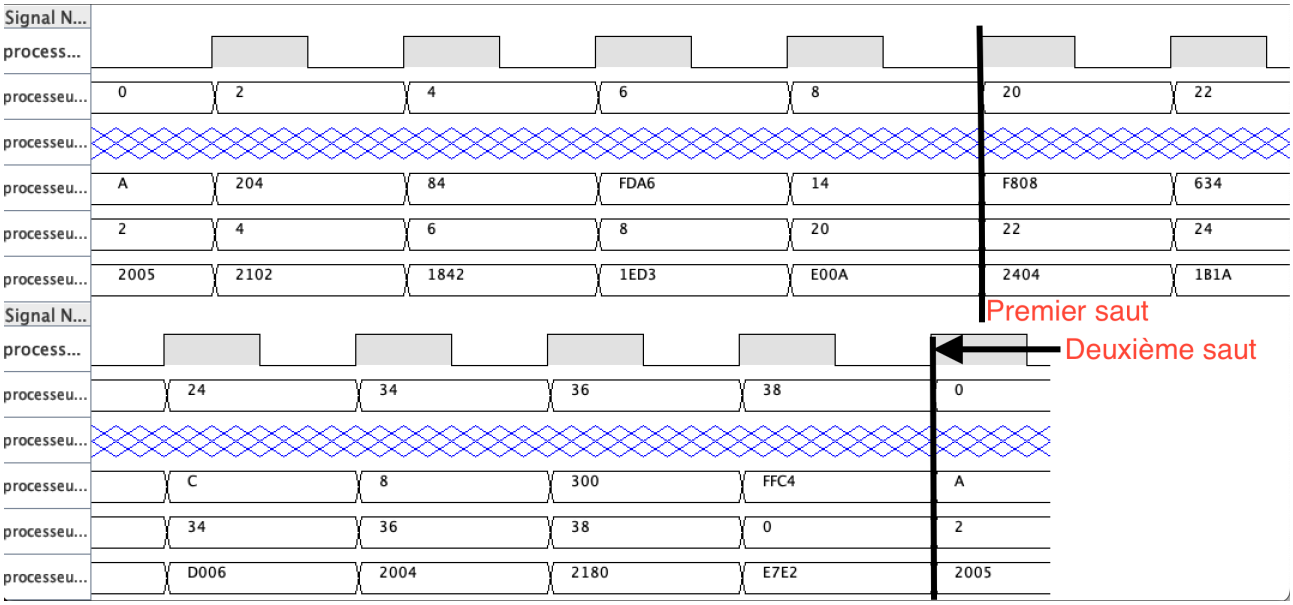
`add r0, r2, r1` : Cette instruction additionne les valeurs de `r2` et `r1` et place le résultat dans `r0`. Son code binaire pourrait être `0xE0801002`.

`and r3, r0` : Cette instruction effectue une opération **ET** entre `r3` et `r0`, et place le résultat dans `r3`. Son code binaire pourrait être `0xE0203000`.

3-

Le lien entre **`PC_pres_i`** et **`PC_fut_o`** réside dans la mise à jour de l'adresse du programme. **`PC_pres_i`** contient l'adresse de l'instruction actuellement en cours d'exécution, et **`PC_fut_o`** contient l'adresse de l'instruction suivante, généralement calculée en ajoutant une valeur constante (comme 4 pour un processeur 32 bits) ou modifiée en cas de branchement (comme un saut). Après chaque cycle d'horloge, **`PC_fut_o`** devient la nouvelle valeur de **`PC_pres_i`**, permettant ainsi au programme de progresser.

Question 4



Question 5

Oui, le programme effectue bien un saut conditionnel. Cela s'explique par le fait que le résultat de l'instruction SUB r2, r3, r4 est égal à zéro, ce qui active le drapeau Z (Z = 1). L'instruction BEQ Label_saut_2 (code machine d006) est alors exécutée, ordonnant un saut conditionnel vers l'adresse spécifiée (0x34). Ce saut dépend strictement de la condition sur le drapeau Z, qui doit être à 1 pour que l'exécution ait lieu.

Question 6

L'offset dans les instructions de saut conditionnel et inconditionnel est une valeur signée. Cela signifie qu'il peut être positif ou négatif.

- Saut vers l'avant : Si l'offset est positif, le programme saute à une adresse plus loin dans le code.

- Saut vers l'arrière : Si l'offset est négatif, le programme retourne en arrière, comme pour une boucle.

Cela permet au programme de se déplacer facilement dans les deux directions, ce qui est utile pour contrôler le flux d'exécution. Voilà pourquoi l'offset est signé

Question 7

Instruction	Valeur en Hexadécimale
e00a	0xa
d006	0x6
E7e2	0x7e2