Q1: Rôle du signal stall\_i

Sur le schéma, le signal stall\_i semble être utilisé pour contrôler le flux d'instructions dans le pipeline. Si ce signal est activé (valeur haute), il indiquerait probablement que le pipeline doit arrêter de traiter de nouvelles instructions jusqu'à ce que la condition de stall soit résolue.

Q2: Signification des signaux rs1\_add\_w et rs2\_add\_w

Les signaux rs1\_add\_w et rs2\_add\_w représentent les adresses des registres sources qui sont lus pendant l'étape d'exécution (EXE) d'une instruction. Ces adresses sont probablement utilisées pour récupérer les valeurs des opérandes pour l'exécution d'une instruction.

Q3: Signification du signal rd\_add\_w

Le signal rd\_add\_w représente l'adresse du registre de destination où le résultat de l'exécution d'une instruction doit être écrit pendant l'étape de write-back (WB). C'est l'adresse où le résultat final d'une opération est stocké.

Q4: Génération actuelle du signal stall\_w

Dans le schéma, stall\_w est généré et semble être connecté à plusieurs multiplexeurs et d'autres composants logiques. Ce signal est probablement dérivé des conditions qui nécessitent un arrêt du pipeline, comme des conflits de données ou des attentes de mémoire.

Q5: Effet de l'activation de stall\_w

Lorsque le signal stall\_w est actif, le pipeline cesse de traiter les instructions subséquentes. Cela permet de résoudre les conditions de dépendance ou d'attendre que les ressources nécessaires deviennent disponibles.

Q6: Type de dépendance gérable par le pipeline avec stall\_w

Si stall\_w est généré en utilisant les signaux d'adresses des registres (rs1\_add\_w, rs2\_add\_w, et rd\_add\_w), cela indique que le pipeline est capable de gérer les dépendances de données en détectant les situations où une instruction en attente d'exécution nécessite des données qui sont encore en cours de traitement ou de modification par des instructions précédentes. Cela pourrait gérer des dépendances telles que:

RAW (Read After Write): Un stall est nécessaire si une instruction doit lire une donnée qui n'a pas encore été écrite par une instruction précédente.

WAR (Write After Read) et WAW (Write After Write): Bien que moins courants dans les pipelines simples, ces types de dépendances peuvent également nécessiter des stalls si les instructions sont exécutées hors de l'ordre ou si plusieurs instructions modifient les mêmes données.

Q7 :

addi r5,r0,#0 🡪 000000000000 r0 000 r5 0010011 🡪 0000 0000 0000 0000 0000 0010 1001 0011

🡪0x00000293

addi r6,r0,#0 🡪 000000000000 r0 000 r6 0010011 🡪 0000 0000 0000 0000 0000 0011 0001 0011

🡪0x00000313

addi r7,r0,#0 🡪000000000000 r0 000 r7 0010011🡪 0000 0000 0000 0000 0000 0011 1001 0011

🡪0x00000393

ori r5,r5,2 🡪 000000000010 r5 110 r5 0010011 🡪0000 0000 0010 0010 1110 0010 1001 0011

🡪0022E293

ori r6,r6,3 🡪 000000000011 r6 110 r6 0010011 🡪0000 0000 0011 0011 0110 0011 0001 0011

🡪0x00336313

ori r7,r7,4 🡪 000000000100 r7 110 r7 0010011 🡪0000 0000 0100 0011 1110 0011 1001 0011

🡪0x0043E393

add r28,r5,r5 🡪 0000000 r5 r5 000 r28 0110011 🡪 0000 0000 0101 0010 1000 1110 0011 0011

🡪0x00528E33

add r29,r6,r6🡪 0000000 r6 r6 000 r29 0110011 🡪 0000 0000 0110 0011 0000 1110 0111 0011

🡪0x00630E83

add r30,r7,r7🡪 0000000 r7 r7 000 r30 0110011 🡪 0000 0000 1000 0100 0000 1111 0011 0011

🡪0x00738F33

addi r0,r0,#0 🡪 000000000000 r0 000 r0 0010011🡪 0000 0000 0000 0000 0000 0000 0001 0011

🡪0x00000013

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Q8/

Le code donne ligne par ligne :

R5 = 0

R6 = 0

R7 = 0

R5 = 2

R6 = 3

R7 = 4

R28 = 2\*R5 = 2\*2 = 4

R29 = 2\*R6 = 2\*3 = 6

R30 = 2\*R7 = 2\*4 = 8

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

Nous pouvons voir que les registres 28 29 30 et 31 ne sont pas modifié

Il y a un problème de gestion du pipeline comme nous pouvons le voir dans le tableau ci-dessous :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 |  |  |  |
| ADDI R5,R0,0 | FE | DE | EX | MEM | WB |  |  |  |  |  |  |  |  |  |
| ADDI R6,R0,0 |  | FE | DE | EX | MEM | WB |  |  |  |  |  |  |  |  |
| ADDI R7,R0,0 |  |  | FE | DE | EX | MEM | WB |  |  |  |  |  |  |  |
| ORI R5,R5,2 |  |  |  | FE | DE | EX | MEM | WB |  |  |  |  |  |  |
| ORI R6,R6,3 |  |  |  |  | FE | DE | EX | MEM | WB |  |  |  |  |  |
| ORI R7,R7,4 |  |  |  |  |  | FE | DE | EX | MEM | WB |  |  |  |  |
| ADD R28,R5,R5 |  |  |  |  |  |  | FE | DE | EX | MEM | WB |  |  |  |
| ADD R29,R6,R6 |  |  |  |  |  |  |  | FE | DE | EX | MEM | WB |  |  |
| ADD R30,R7,R7 |  |  |  |  |  |  |  |  | FE | DE | EX | MEM | WB |  |
| ADDI R0,R0,#0 |  |  |  |  |  |  |  |  |  | FE | DE | EX | MEM | WB |

Il existe 2 solution : -Ajouter un NOP pour éliminer les dépendances

-Ou bloquer le FETCH

Q11/

Pour coder un NOP nous pouvons faire la commande : ADDI R0,R0, #0

Q12/

Il s’agit de dépendance entre les DE et WB nous n’avons pas encore pu renvoyer la nouvelle valeur via le WB quand on décode, que ce soit entre ADDI et ORI ou ORI et ADD.

Nous aurons donc besoin de 2 NOP 1 avant chaque changement de commande (ADDI -> ORI) et (ORI -> ADD) en effet le NOP va décaler le FETCH pour les instructions suivantes donc pas de problème pour R6 et R7

Q13/

Une image contenant texte, capture d’écran, Police, nombre

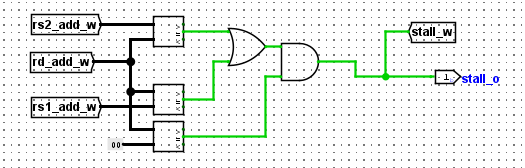
Description générée automatiquement

Voici les nouvelles commandes en ayant ajouté les NOP.

Q14/

WB ,DE, et MEM était concerné par les dépendances

Q15/



Q16/