UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

PLAN DE VÉRIFICATION

Architecture et organisation des
Ordinateurs
APP 3

Présenté à
Marc-André Tétrault

Présenté par
Gabriel Lamontagne – lamg0502
Kevin Rondeau – ronk2602

Sherbrooke – 19 janvier 2024

TABLE DES MATIÈRES

1. Plan de vérification **1**

1.1 Spécifications sommaires 1

1.1.1 Figure de l'architecture modifiée 2

1.1.2 Description des modifications 3

1.2 Plan 4

1. PLAN DE VÉRIFICATION

1.1 SPÉCIFICATIONS SOMMAIRES

Voici les nouvelles instructions implémentées :

- lwv rs, rt : load word vector
- addv rd, rs, rt : add vector
- vmin rs, rt : regarde un vecteur et enregistre la valeur la plus petite dans une variable. Par exemple, vmin $t0, $v1 où :
    - $v1 = 3 4 1 2
    - La variable résultante $t0 sera : 1

1.1.1 FIGURE DE L'ARCHITECTURE MODIFIÉE

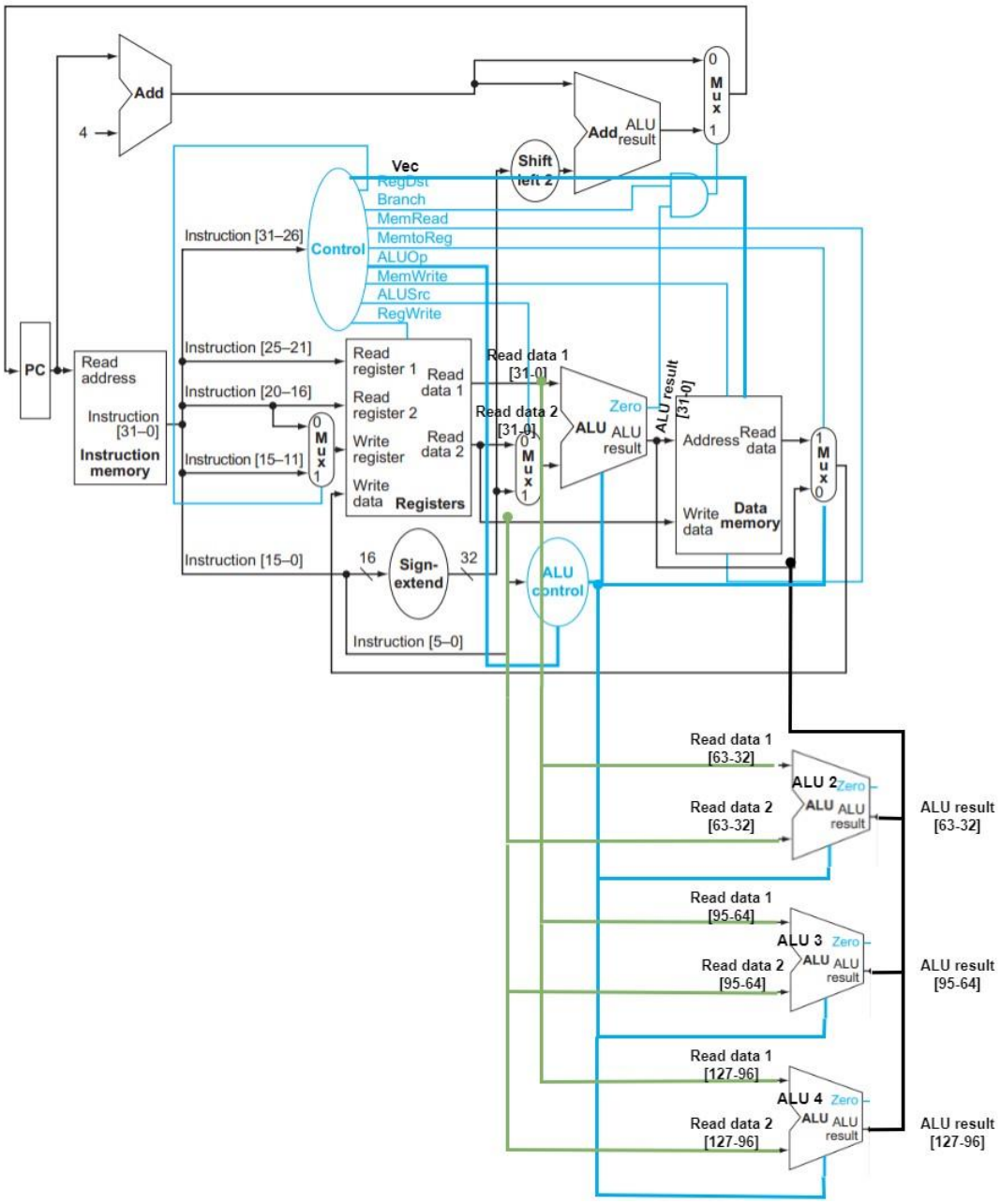

Figure 1 : Nouvelle architecture du CPU

1.1.2 DESCRIPTION DES MODIFICATIONS

Afin d'accommoder les nouvelles instructions SIMD, la mémoire de donnée fut modifiée pour écrire et lire des données de 128 bits. Si on voulait lire un vecteur en mémoire, un signal était envoyé au module de mémoire de données et celui-ci allait voir l'adresse pointé en entrée, et sortait la valeur à cette adresse ainsi que les valeurs aux trois adresses suivantes en les combinant dans le signal de 128 bits. Autrement, les bits 127 à 32 étaient remplis de 0 et un mot unique pouvait être mis en sortie dans les bits 31 à 0. Le registre devait également écrire des données de 128 bits, et mettre en sortie des données lu de 128 bits. Dans celui-ci cependant, nous avons remplacé les registres réservés à $s0 jusqu'à $s7 pour des registres utilisant 128 bits de mémoire plutôt que 32 afin de les réserver pour des vecteurs de 4 mots. Ainsi, à la lecture des entrées du registre, le registre savait s'il devait écrire ou lire dans la mémoire de 32 ou 128 bits. Afin de traiter ce bus de données plus grand, le module ALU fut répliquer 3 autres fois. Si un mot régulier était utilisé, seule la sortie de l'ALU original (bits 31 à 0) était utilisée mais si on voulait opérer sur un vecteur, on se servait de l'entièreté du bus de 128 bits. C'est également pourquoi, seul l'ALU original est relié à l'entrée Adresse du module de mémoire de données.

1.2 PLAN

Test	Action à faire	Résultats attendus	Résultat atteint
Charger un vecteur dans un registre vide (lwv)	Envoyer une instruction pour charger le vecteur commençant à l'adresse 0($a0) contenant les valeurs 1, 2, 3, 4 dans le registre $v0	Le vecteur à l'adresse 0($a0) est chargé dans le registre $v0 $v0 = « 1 2 3 4 »	[1]
Charger un vecteur dans un registre qui contient déjà des valeurs (lwv)	Envoyer une instruction pour charger le vecteur commençant à l'adresse 0($a0) contenant les valeurs 1, 2, 3, 4 dans le registre $v0	Le vecteur à l'adresse 0($a0) est chargé dans le registre $v0 $v0 = « 1 2 3 4 »	[1]
Valeur minimale dans un vecteur contenant des valeurs différentes (vmin)	Envoyer une instruction pour comparer les valeurs du vecteur $v1 = « 3 4 1 2 »	La valeur résultante devrait être : $v0 = « 1 »	[1]
Valeur minimale dans un vecteur qui contient deux paires de données (vmin)	Envoyer une instruction pour comparer les valeurs du vecteur $v1 = « 3 4 3 4 »	La valeur résultante devrait être : $v0 = « 3 »	[1]
Addition de vecteurs qui contiennent tous deux des valeurs (addv)	Envoyer une instruction pour additionner les vecteurs $v1 = « 3 4 4 1 », $v2 = « 1 2 3 4 ».	Le vecteur résultant devrait être : $v0 = « 4 6 7 5 »	[1]
Addition de vecteurs dont un des vecteurs est null (addv)	Envoyer une instruction pour additionner les vecteurs $v1 = « 3 4 4 1 », $v2 = « 0 0 0 0 ».	Le vecteur résultant devrait être : $v0 = « 3 4 4 1 »	[1]