

Proc-8-bits - Conception d'un processeur avec jeu d'instructions élémentaires - v1.0

David DEVANT - Aurélien TROMPAT

11 février 2019

Table des matières

1	Architecture	2
2	Architecture CPU	2
2.1	Contrôleur (UC)	2
2.1.1	Registre d'instruction	2
2.1.2	Compteur d'adresse	2
2.1.3	Multiplexeur	3
2.1.4	Machine d'état (FSM)	3
2.2	Contrôleur (UT)	4
2.2.1	Registre de données mémoire	4
2.2.2	Registre d'accumulation	4
2.2.3	Unité de calcul (UAL)	4
2.2.4	Flipflop de retenue (Carry)	4
2.3	Mémoire	4
3	Tests	5
3.1	Additionneur N-bits	5
3.2	Flipflop	5
3.3	Multiplexeur à 2 entrées	5
3.4	Unité de calcul	5
3.5	CPU - Programme du PGCD	6
3.6	FSM	6
4	Conclusion	7

Introduction

L'objectif de ce projet est de concevoir un processeur 8 bits disposant d'un jeu de 4 instructions élémentaires : Non-OU, Addition, Stockage mémoire, Saut d'adresse

1 Architecture

L'architecture de notre CPU est imposée par le cahier des charges. Elle se divise en 3 parties :

- Contrôleur (UC) : il est l'unité de contrôle qui génère des signaux internes de commandes.
- Calculateur (UT) : Il effectue des calculs sur des données d'entrée et retourne de résultat vers la mémoire.
- Mémoire : Elle stocke le programme et les variables.

2 Architecture CPU

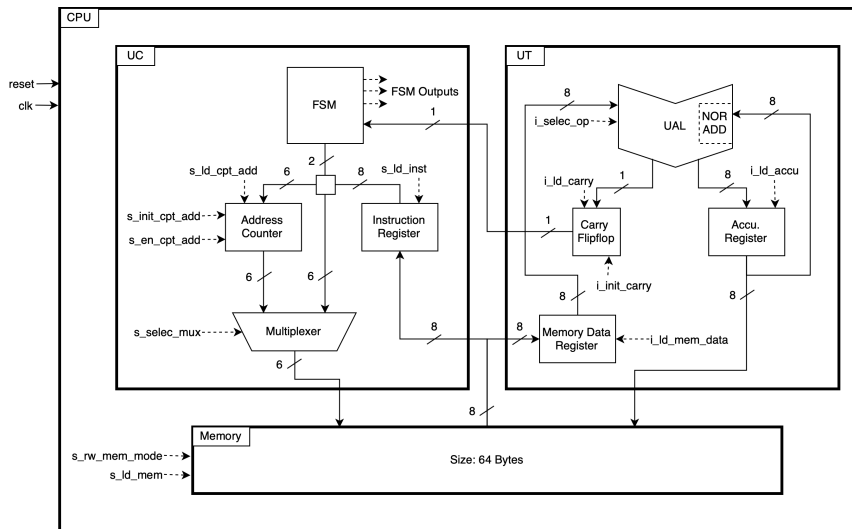


FIGURE 1 – Architecture du CPU (Clock et reset non représenté)

2.1 Contrôleur (UC)

2.1.1 Registre d'instruction

Il permet de stocker l'instruction qui est actuellement traitée par le CPU. Les instructions sur 8 bits sont issues de la mémoire.

2.1.2 Compteur d'adresse

Ce compteur permet de définir l'adresse mémoire de la prochaine instruction que devra exécuter le CPU. Un système de chargement permet de modifier la valeur courante du compteur pour permettre des sauts d'adresse (Instruction

JCC). Une entrée "init" permet de remettre à 0 le compteur pour redémarrer le programme à l'adresse 0.

2.1.3 Multiplexeur

Le multiplexeur est un composant purement combinatoire qui permet de sélectionner la source de l'adresse mémoire à lire/écrire. Si son signal de commande est à '0', alors c'est la sortie du compteur d'adresse qui est connecté à la mémoire. Ceci permet de lire les instructions du programme. Lorsque le signal de commande est positionné à '1', ce sont les 6 bits de poids faibles qui fixent l'adresse. Ceci permet d'effectuer les accès mémoire pour la partie calcul.

2.1.4 Machine d'état (FSM)

La FSM est l'ordonnanceur du CPU. Elle génère 12 signaux qui commandent les différents composants du CPU. Voici son schéma :

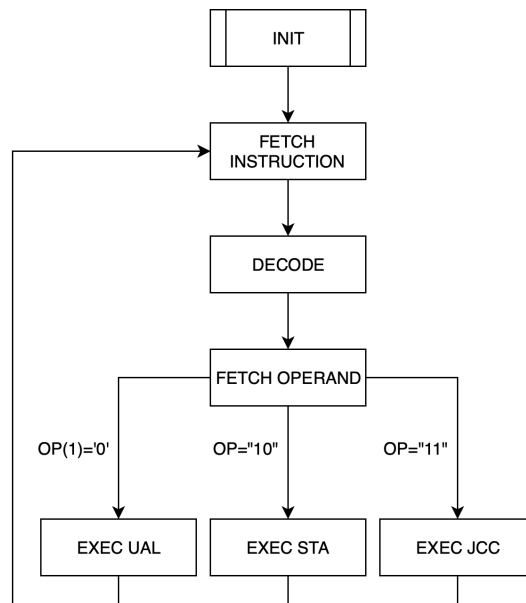


FIGURE 2 – Diagramme de la machine d'état du CPU

- INIT : Phase d'initialisation du CPU, cette étape n'est active qu'après un reset.
- FETCH INSTRUCTION : Chargement de l'instruction indiquée par le compteur d'adresse.
- DECODE : Lit le code opération de l'instruction et anticipe un chargement de mémoire.

- **FETCH OPERAND** : Chargement de la variable pointée par l'instruction de la mémoire vers la partie opérative (UT). Le choix de l'étape suivante est conditionné par le code d'opération.
- **EXEC UAL** (Code OP = "00" ou "01") : Définit l'opération à effectuer sur la partie opérative puis ordonne la sauvegarde du résultat dans le registre d'accumulation.
- **EXEC STA** (Code OP = "10") : Sauvegarde la valeur du registre d'accumulation en mémoire à l'adresse indiquée par l'instruction.
- **EXEC JCC** (Code OP = "11") : Effectue un test sur la valeur de la retenue pour déterminer si le saut d'adresse doit être exécuté. Le saut est effectué en plaçant les 6 bits de poids faible de l'instruction dans le compteur d'adresse.

2.2 Contrôleur (UT)

2.2.1 Registre de données mémoire

Ce registre permet de stocker la dernière valeur lue en sortie de la mémoire. Il est directement lié à l'unité de calcul UAL.

2.2.2 Registre d'accumulation

Ce registre stocke le résultat de la partie UAL. Sa valeur peut être envoyée vers la mémoire pour être sauvegardée ou bien réutilisée pour le calcul suivant.

2.2.3 Unité de calcul (UAL)

Cette unité de calcul dispose d'une entrée de sélection d'opération entre l'addition et le Non-OU. L'opération sélectionnée est appliquée entre les deux valeurs 8 bits placées en entrée et le résultat est retourné vers le registre d'accumulation. Une seconde sortie permet de récupérer la retenue sur 1 bit pour être dirigée vers la flipflop.

2.2.4 Flipflop de retenue (Carry)

Cette bascule de type flipflop permet de stocker l'état de la retenue du dernier calcul de l'UAL. Sa sortie est utilisée par la FSM pour déterminer si le saut d'adresse de l'instruction JCC doit être effectué.

2.3 Mémoire

Cette mémoire de 64 octets est une mémoire simple accès avec un port de lecture et un second d'écriture. Elle est adressée sur 6 bits et manipule des valeurs sur 8 bits.

3 Tests

Chaque composant a été testé indépendamment à l'aide de "testbench". Une fois fonctionnels, nous les avons implémentés dans le top layer "CPU". Pour valider le projet, nous avons écrit les deux programmes de test vu en cours dans des fichiers "*.data". Ces fichiers sont chargés dans la mémoire juste avant la simulation. En observant les derniers accès mémoire du programme, on observe que les résultats sont cohérent avec les résultats attendus.

3.1 Additionneur N-bits

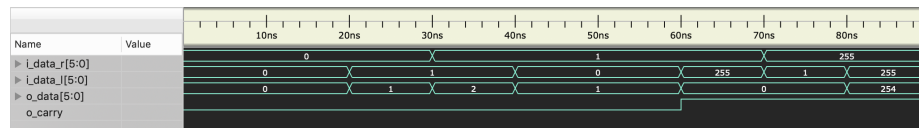


FIGURE 3 – Diagramme temporel de l'additionneur N-bits

3.2 Flipflop

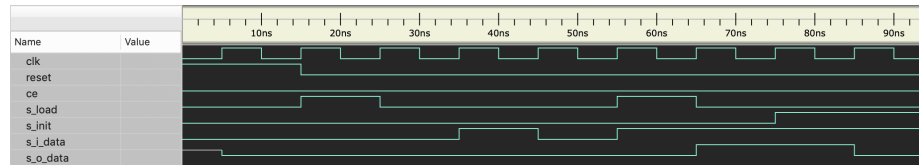


FIGURE 4 – Diagramme temporel de la flipflop de retenue

3.3 Multiplexeur à 2 entrées

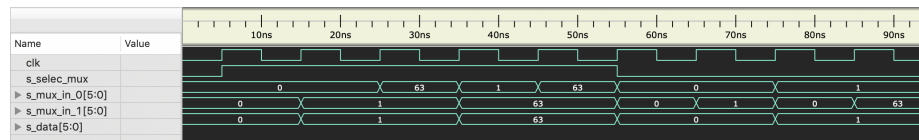


FIGURE 5 – Diagramme temporel du multiplexeur

3.4 Unité de calcul

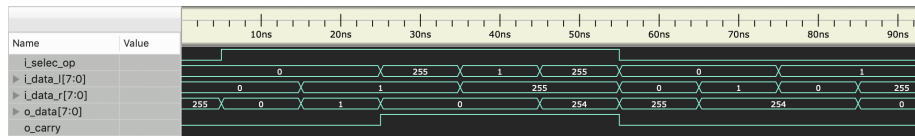


FIGURE 6 – Diagramme temporel de l'unité de calcul UAL

3.5 CPU - Programme du PGCD

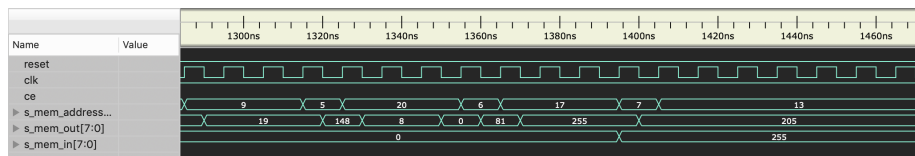


FIGURE 7 – Diagramme temporel du CPU avec programme de PGCD

3.6 FSM

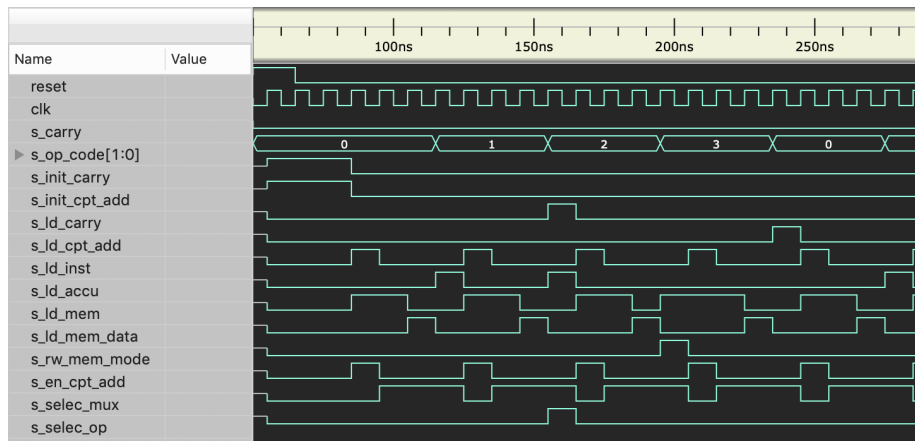


FIGURE 8 – Diagramme temporel de la FSM

4 Conclusion

Ce projet nous a fait découvrir les fonctionnements internes d'un processeur ainsi que les problématiques d'accès mémoire. L'implémentation en VHDL nous a permis de se remémorer les bonnes pratiques des langages de description. De plus, la réutilisation des nos anciens composants VHDL nous a montré qu'il était important de bien structurer et bien documenter nos fichiers pour être plus efficaces dans nos futurs projet.