Rapport d’avancement de Projet

VLSI

# BISLIEV Magomed-Salakh

# YANG Zixiao

2022/2023

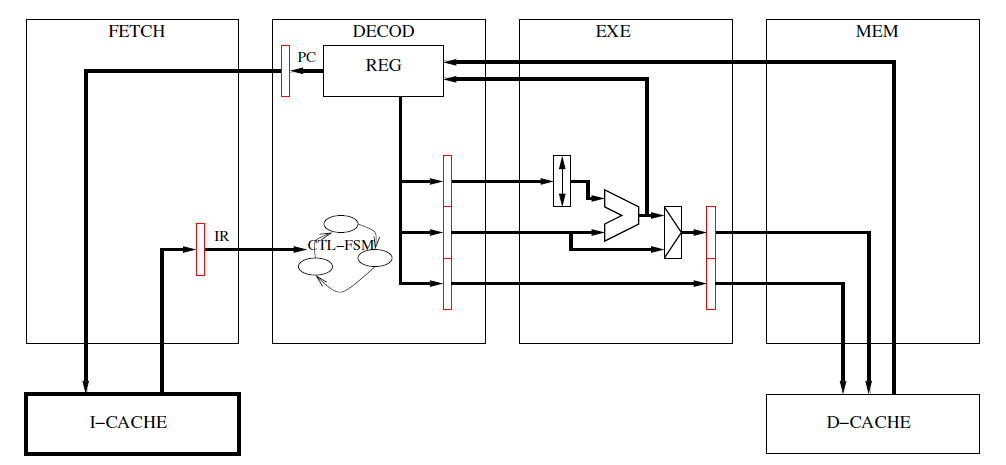
# Sommaire

# Introduction

Dans le cadre de ce projet nous réalisons un processeur ARM qui comporte 4 étages IFETCH, DECODE, EXE et MEM.

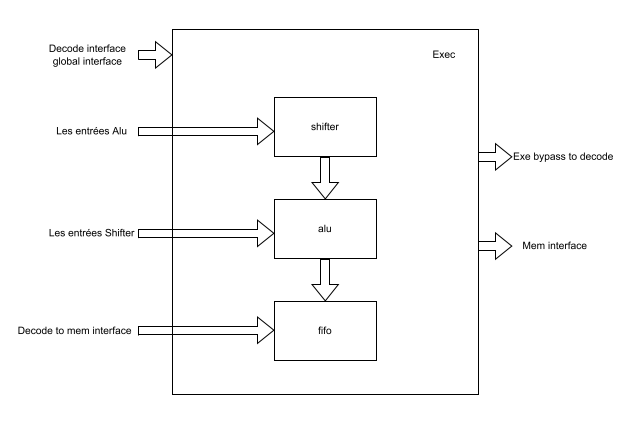
* IFETCH : récupération de l’instruction
* DECOD : décodage de l’instruction
* EXE : exécution de l’instruction
* MEM : Accès mémoire pour stocker ou charger une donnée

Nous avons réussi à réaliser l’étage EXE et



# Étage exe

L’étage exe étant chargé d’exécuter les instructions, il doit réaliser des opérations élémentaires et pour cela il contient deux blocs qui sont l’arithmetic logic unit (ALU) et le shifter.



C’est le schéma bloc du Exe, je pense qu’il faut rajouter qlq trucs

D’après ce schéma bloc, on peut savoir la relation et les ordres entre alu shifter et fifo, tout d’abord on utilise shifter pour décaler les valeurs, après on utilise Alu pour calculer l’opérand qu’on veut, à la fin on prend les résultats dans notre fifo pour faire les suivantes

1. Arithmetic logic unit

L’Alu est le composant qui nous permettra de réaliser les 4 opérations de base : ADD avec retenue, AND, OR et XOR. Mais afin de sélectionner l’opération voulue entre les deux opérandes de 32 bits, nous utilisons une commande sur deux bits. Les opérations AND, OR et XOR sont très simples à implémenter à l’aide des opérateurs qui sont déjà mis à disposition. Cependant, pour l’opération d’addition, nous avons utilisé un Full Adder en l’instanciant 32 fois. Lors de la synthèse nous avons constaté que l’instanciation avec un seul “generate (de 0 à 31)” génère une erreur sur les retenues car les retenues 1 à 31 ne sont pas assignées. En effet, nous avons dû réaliser une première instanciation pour produire le résultat de rang 0, ensuite l’instanciation avec un generate de 1 à 30 et une dernière instanciation pour produire le résultat de rang 31.

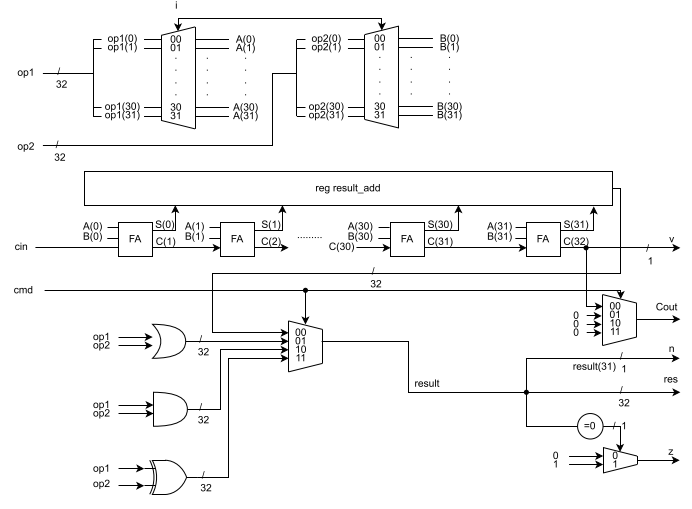


Schéma des opérations réalisés par l’ALU

Dans ce schéma bloc, nous effectuons différentes opérations selon différentes situations de cmd

Lorsque cmd = 00, nous calculons l’opération Add de 32 bit par adder\_1bit et la sortie cout

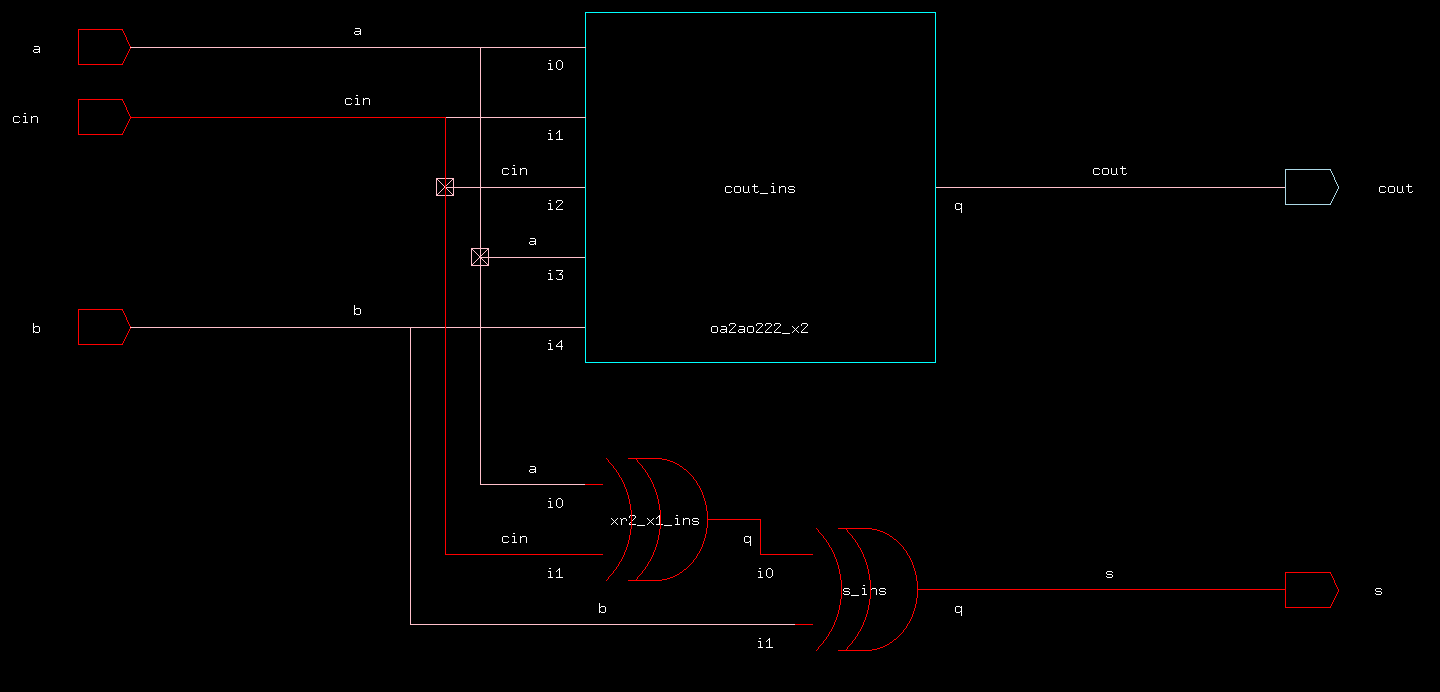
Quand cmd=01, on calcule l’opération Or

Quand cmd=10, on calcule l’opération And

Lorsque cmd=11, on calcule l’opération Xor

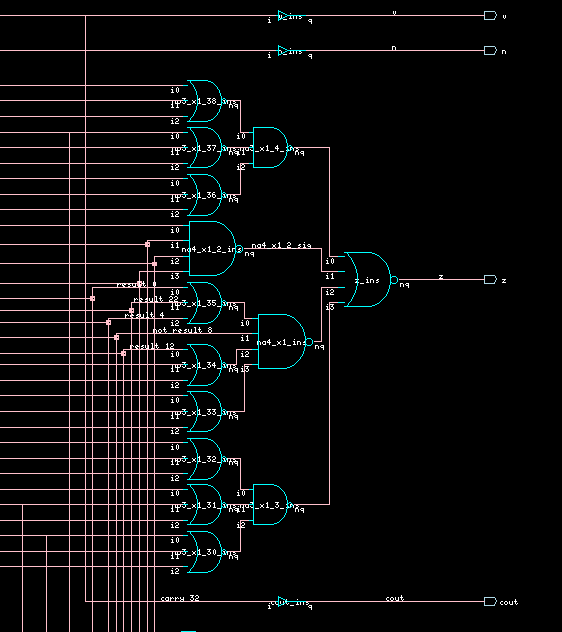
c’est comme ce que t’as écirt sur en haut

| cmd | Opération |
| --- | --- |
| 00 | ADD |
| 01 | OR |
| 10 | AND |
| 11 | XOR |



Résultat de synthèse pour le fulladder

vérifier que oa2ao222\_x2 est cohérent



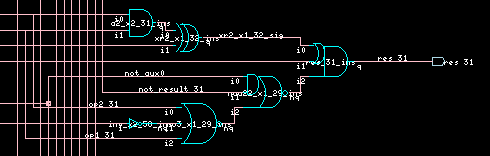
Résultat de synthèse pour l’alu (zoom sur la génération des flags czn et v)

c : retenue de sortie

z : résultat nul

n : résultat négatif

v : dépassement de capacité

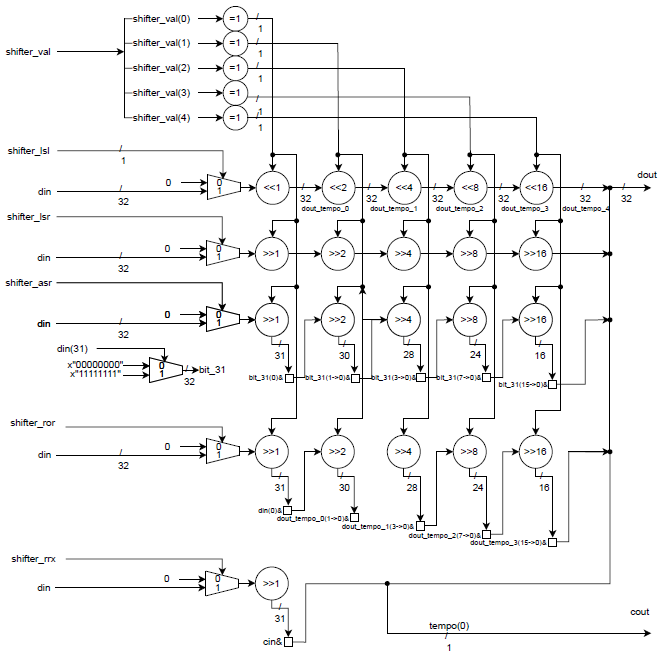


Résultat de synthèse pour l’alu (zoom sur le bit 31 du vecteur res)

Le vecteur res correspond à la sortie de l’alu sur 32 bits et le schéma en portes logiques des bits de rang inférieur reste identique à celui représenté sur la figure …

1. Shifter

Un autre composant essentiel pour notre processeur ARM est le shifter. En effet, ce-dernier nous permet de réaliser des décalages et rotations. La technique employée consiste à réaliser des décalages ou des rotations progressifs. La valeur de décalage étant définie par une valeur binaire sur cinq bits, nous pouvons décomposer l’opération en des décalages de 2^N avec N=0, 1, 2, 3 et 4. Ainsi, chaque valeur de décalage comprise entre 0 et 31 peut être réalisée. On comprend donc que nous réalisons des concaténations de 0 ou 1 en fonction de s’il s’agit d’un décalage logique ou arithmétique. Pour les décalages logiques : Logic shift left(LSL) et Logic shift right(LSR) nous réalisons simplement des concaténations de 0. Cependant, dans le cas d’un décalage arithmétique vers la droite : arithmetic shift right(ASR) nous devons tenir compte du bit de signe en réalisant l’extension de ce-dernier. Pour la rotation nous utilisons directement les bits de poids faible de notre opérande. Le principe est présenté sur la figure ci-dessous.



D’après ce schéma bloc, nous effectuons différentes opérations selon différentes situations de command lsl,lsr,arsr ror et rrx

On utilise la décalage de 1,2,4,8,16 pour chaque bit de valeur de décalage vaut 1.

Le bit din(31) est utilisé pour réaliser une extension de signe dans le cas où l’on souhaite réaliser un ASR.

1. fifo

FIFO(First In First Out) littéralement en français premier entré premier sorti. Il s'agit d'une méthode de gestion des stocks dont l'objet est de faire sortir les marchandises et matières premières par ordre d'entrée en stock. Les entreprises utilisant des denrées périssables pour leur production ont recours à la méthode du premier entré premier sorti. Dans notre étage EXE, nous devons vérifier si les fifos de sortie ne sont pas full et les fifos d’étage DEC ne sont pas vide, pour traiter les informations en EXE.

On sait que pour un processeur pipeliné chaque étage possède un matériel, ici le matériel est représenté par une pile de type First In First Out. En effet, cette dernière nous permet de séparer les étages et de stocker

faut ecrire un peu , bon courage magomed, jsp comment ecrire:)

# Étage Decode

L’étage decode est l’étage le plus complexe du processeur car ce dernier est chargé de décoder les instructions de l’architecture ARM… . En effet, il existe différents types d’instructions et ces dernières utilisent des opérandes sur 32 bits, il convient d’utiliser des registres pour ces opérandes et pour cela nous avons utilisé un bloc appelé reg. Ce bloc contient

§ 3 ports de lecture numérotés de 1 à 3,

§ 2 ports d’écriture, le numéro 1 correspond à exec et est donc prioritaire,

§ 2 ports d’invalidation car une instruction peut produire 2 résultats,

§ 4 *flags* et leurs 2 bits de validité,

§ PC, son bit de validité et sa commande de +4

Pour le banc de registre nous avons décidé de réaliser un process pour l’écriture dans les registres et la gestion des bits de validité associés, la mise à jour des flags et la gestion des bits de validité associés et la gestion de pc. Cependant, pour la lecture nous avons décidé de la réaliser en concurrent. Pour l’écriture lorsqu’un registre est identifié comme une destination, ce dernier est invalidé, pour cela une solution est d’utiliser un vecteur invalid dont le bit de rang n indique s’il est égal à 1 que le registre n est invalidé. Afin de gérer la priorité entre le port 1 et 2, nous avons crées 2 autres vecteurs wadr1 et wadr2, lorsque l’on souhaite écrire dans le registre n via le port 1 il faut que wadr1(n) soit à 1, cependant lorsque l’on veut écrire dans le registre n via le port 2 il faut que wadr2(n) soit à 1 et wadr1(n) à 0.

