COMPTE RENDU INTERMEDIAIRE 2

PROJET RISC-V



1 TABLE DES MATIERES

2	Introduction	2
3	Compilateur C pour RV32I	2
4	Ecriture des architectures et leurs test	3
5	Conclusion	4
Figu	re 3.1 - Résultat de la compilation du code assembleur à droite	2
Figure 4.1 - Extrait du code RTL du bloc imm_gen		
_	Figure 4.2 - Résultat de la simulation du bloc imm, gen	



2 Introduction

Dans cette deuxième partie du projet, plusieurs étapes ont été réalisées. Le but ici est de faire un petit récapitulatif des avancements. L'écriture des architectures en SystemVerilog sont pratiquement toutes terminées et testées, il ne reste plus qu'à réaliser une simulation sur le système au complet. Nous avons pour objectif de pouvoir exécuter (en simulation) des codes C. Pour ce faire nous avons trouvé un site qui va compiler le C en langage assembleur avant d'utiliser notre projet informatique de premier semestre (modifié) pour compiler l'assembleur en code binaire correspondant au jeu d'instruction de notre processeur.

3 COMPILATEUR C POUR RV32I

Afin de pouvoir réaliser des testbenchs plus complexe il était nécessaire de pouvoir avoir un compilateur pouvant traduire du code C vers un code machine pouvant être exécuter pour notre processeur. Pour cela nous nous sommes tournés vers un site permettant de compiler du code vers de l'assembleur : https://godbolt.org/. Cependant nous avons dû faire face à plusieurs soucis. Le premier étant que le compilateur utilisé, le RISC-V rv32gcc gcc 10.2.0 ne semble pas avoir d'option pour seulement compiler pour un jeu d'instruction RV32I sans extension. De plus le site ne nous fournis pas non plus un code machine spécifiquement exécutable pour notre processeur.

Nous nous sommes alors tournés vers le compilateur réalisé lors du projet d'informatique de début d'année. L'objectif ici était de modifier les expressions régulières ainsi que le code de la fin du livrable 2 et le livrable 3 afin de pouvoir lire le code assembleur fournis par le site et l'exporté en code machine au format ASCII.

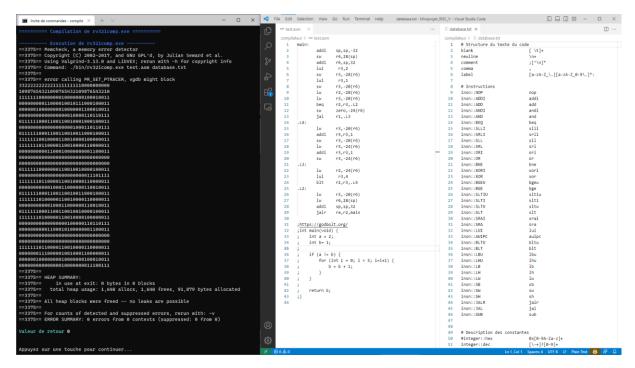


Figure 3.1 - Résultat de la compilation du code assembleur à droite

Notre compilateur nous permet également de rajouter des opérations "vides", des instructions *NOP* après un branchement pour éviter que certaines instructions soient ignorées ou prise en compte alors qu'on prend le branchent.

Ecriture des architectures et leurs test



Nous songeons aussi à automatiser le remplacement de certaines instructions pouvant être générer lors de la première compilation, tel que des *LI* ou *MV* par des *ADDI* ou *ADD*.

4 ECRITURE DES ARCHITECTURES ET LEURS TEST

L'écriture de la plupart des architectures a été réalisée ces deux dernières semaines. Pour chaque bloc écrit, un bench fut réalisé pour le tester. Voici par exemple un code RTL d'une partie du processeur et son testbench :

```
imm_gen imm_gen1(
    .clk(clk),
    .rst(rst),
    .sig(sig),
    .imm_in(imm_in),
    .imm_out(imm_out)
);
initial $timeformat ( -9, 1, " ns", 12 );

// Clock and Reset Definition
'define PERIOD 10
initial
begin
    clk = 1'b1;
    reset = 1'b0;
    end

always
    #('PERIOD/2) clk = ~clk;

initial forever
begin
    sig = 3'b001;
    imm_in = 32'hffffffff;
#100
    sig = 3'b010;
    #100;
#100
    sig = 3'b100;
#100;
#100;
end

// Automatic checker to compare Filter value with expected Output
```

Figure 4.1 - Extrait du code RTL du bloc imm_gen

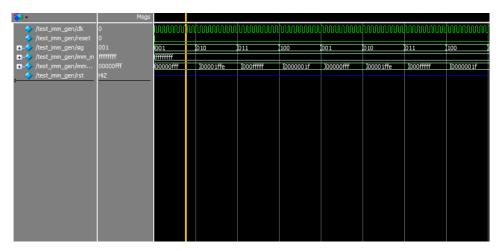


Figure 4.2 - Résultat de la simulation du bloc imm_gen

Il ne reste plus qu'à tester cette semaine le processeur de manière globale avec de simples instructions et avec un jeu en entier fourni par le compilateur. Le but est, dans un premier temps, de tester individuellement des instructions ou jeux d'instructions qui peuvent être problématiques (ex : dépendance de données, jump etc...). Dans un second temps nous allons pouvoir faire exécuter des programmes complets, pour ainsi vérifier qu'à chaque étape de l'exécution, jusqu'à la fin, on obtienne les valeurs attendues en mémoire.

Conclusion



5 CONCLUSION

Nous sommes assez satisfaits de l'avancement du projet. Les prochaines semaines seront essentiellement consacrées au test de notre microprocesseur. Nous n'avons pas vraiment d'idée sur le temps que ça va prendre (ça peut marcher du premier coup comme nécessiter de gros ajustements). Si la phase de test est rapide nous pouvons commencer une implémentation ASIC pour notre processeur sinon on se contentera de la simulation.