SORBONNE UNIVERSITÉ

M1 SESI

Modélisation SystemC d'un RISC-V pipeliné

Rapport d'avancement
dans le cadre du projet PSESI

Etudiants:
M. Timothée Le Berre
M. Louis Geoffroy Pitailler
M. Kevin Lastra

Encadrante :
Mm. Daniela Genious

Nous souhaitons remercier
Mme. Daniela Genius, M. Pirouz Bazargan Sabet et M. Franck Wajsbürt
qui nous ont énormément aidé lors de la réalisation de ce projet.

Table de Matières

1 Introduction **2**
1.1 Historique de l'architecture RISC : 2
1.2 Pourquoi choisir ce projet ? 3

2 Objectifs **5**

3 Situation actuelle du projet **6**
3.1 Résumé graphique 6
3.2 Validation 6

4 Problèmes rencontrés lors de la conception **7**

5 Objectifs restants **7**

1 Introduction

1.1 Historique de l'architecture RISC :

En 1971, Intel sort son premier microprocesseur, l'Intel 4004 basé sur une architecture 4 bits CISC (Complex Instruction Set Computing).

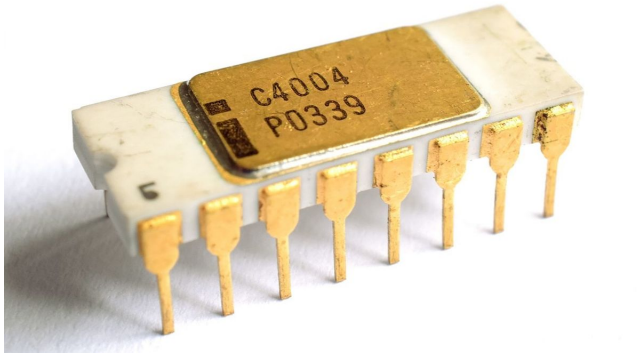


Figure 1: Intel 4004, source : https://www.lesnumeriques.com/cpu-processeur/l-intel-4004-premier-processeur-du-fondeur-fete-aujourd-hui-ses-50-ans-n171113.html

Les Processeurs CISC ont largement domié le marché jusque dans les années 80 où une nouvelle architecture va faire son apparition : l'architecture RISC.
Les architectures CISC sont beaucoup plus complexes que les risques, en effet ces derniers implémente des fonctions très complexes en matériel. On peut par exemple citer l'Intel 8086 qui implémente matériellement des instructions permettant de faire des comparaisons entre des chaines de caractères.
Les architectures RISC au contraire implémente uniquement des fonctions basiques et simple de manière matériel, la philosophie du RISC étant en effet de laisser les taches complexes au compilateur.
RISC était à l'origine un projet mené par David Patterson à l'Université de Berkely en Californie entre 1980 et 1984.
Cette architecture va vite montrer de gros avantages par rapport à l'architecture CISC et de nombreux projets vont se développer en se basant dessus.
En 1981, le MIPS (Microprocessor without Interlocked Pipeline Stages) -qui se base sur une architecture de type RISCV- fait son apparition à l'Université de Standford.
La technologie MIPS va être commercialisée à partir de 1984 et sa première implémentation, le R2000, va devenir l'un des processeurs les plus utilisés pour l'embarqué.

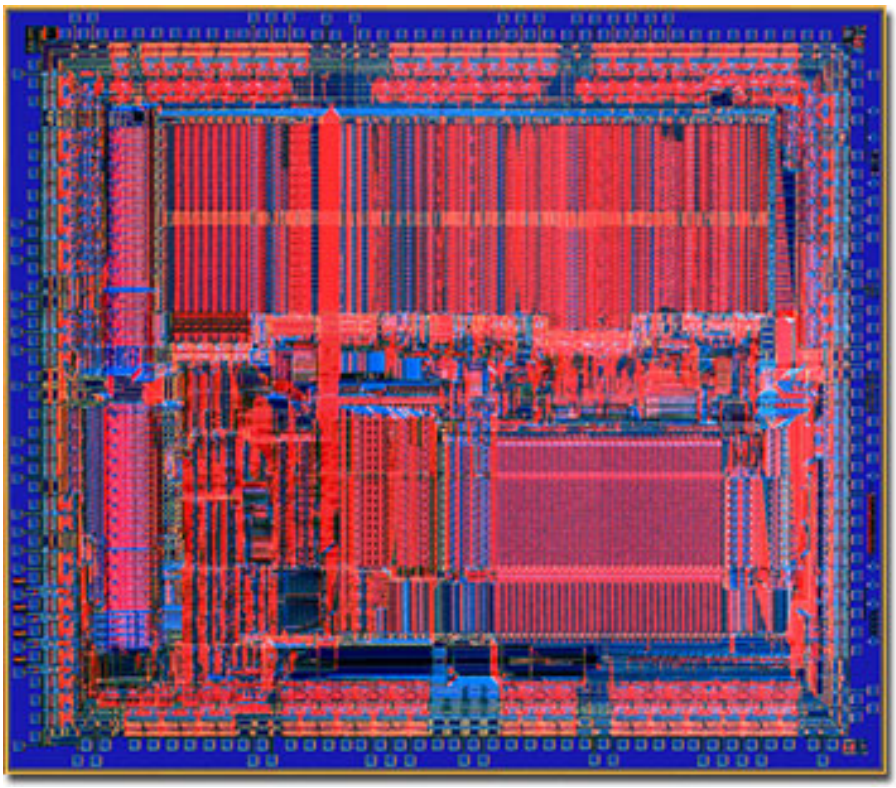


Figure 2: Intel 4004, source : https://www.cpushack.com/MIPSCPU.html

Grâce à sa simplicité de compréhension et son efficacité, le MIPS s'impose comme un standard dans l'enseignement de l'architecture processeur, c'est pourquoi le Lip6 l'a choisi comme base de son enseignement.
Néanmoins, cette architecture a fait son temps et commence peu à peu à se montrer trop vieille. C'est pourquoi il a été décidé de changer la base des cours de Sorbonne Université utilisant en utilisant l'architecture RISC-V, une architecture libre, facile à implémenter, proche du MIPS et surtout utilisé à l'heure actuelle.

1.2 Pourquoi choisir ce projet ?

Au cours de notre premier semestre de M1 SESI, nous avons eu l'occasion d'implémenter une architecture ARMv2a en VHDL. Cela nous a énormément plu et lorsque Mme. Genius a proposé une implémentation d'une architecture RISC-V en systemC [2] nous avons tout de suite postulé afin de pouvoir participer à ce projet.

Ce projet nous a permis de nous familiariser avec le jeu d'instruction RISC-V et avec le langage systemC [2]. RISC-V étant l'une des implémentations de RISC les plus importantes - aux côtés de l'architecture ARM -, il est très intéressant d'en étudier son fonctionnement.
Ne voulant pas simplement réaliser une architecture scalaire et voulant aller plus loin que ce que nous avions déjà eu l'occasion de faire en VLSI avec l'architecture ARM, nous avons décidé d'avancer rapidement sur le projet dans le but de finir début mars l'implémentation scalaire et d'ensuite pouvoir nous concentrer sur une implémentation superscalaire SS2 à pipeliné à 5 étages.

Le SS2 désigne un super-scalaire de 2 étages où les étages EXE, MEM et WBK sont dupliqués. Il s'agit d'une architecture imaginé par Mr. Pirouz Bazargan. Nous l'avons étudié dans le cadre de notre UE ARCHI au 1er

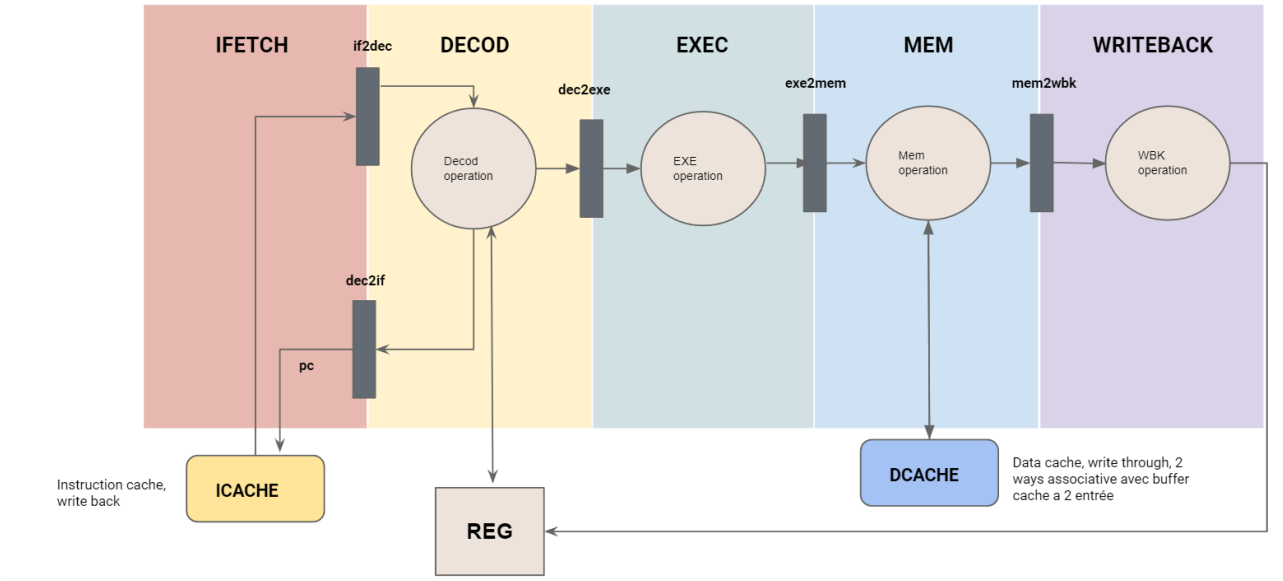
semestre et nous souhaitions donc l'implémenter dans notre design.

Néanmoins, après discussion avec notre encadrante, il a été décidé de commencer d'abord par l'ajout d'une partie Kernel à notre design et d'ensuite passer à l'implémentation SS2 s'il nous restait du temps.

2 Objectifs

L'objectif premier de notre projet était d'implémenter une architecture RISC-V 32 bits pipelinée à 5 étages sans extension, dans le but de remplacer l'architecture MIPS du Lip6. Les étages de notre implémentation sont les mêmes qu'une MIPS R3000, à savoir :

- IFETCH
- DECODE
- EXECUTE
- MEMORY
- WRITE-BACK



Le langage imposé pour cette réalisation est systemC [2]. Notre implémentation devait donc être proche de l'architecture MIPS que nous avons étudié en cours, d'où l'implémentation sous forme d'un pipeline 5 étages. Une fois l'implémentation terminée, nous devions également mettre en place une plateforme de TP.

Finalement il a été décidé que la mise en place de la plateforme de TP n'était pas prioritaire et que l'implémentation de la partie Kernel primait.
Notre objectif est donc de mettre en place une architecture RISC-V scalaire avec partie Kernel et s'il nous reste du temps d'implémenter un SS2 et enfin de mettre en place une plateforme de TP.

3 Situation actuelle du projet

À l'heure où nous écrivons ce rapport, nous avons fini la partie scalaire et le nettoyage du code en accord avec les conventions que nous avons pris pour le nom des signaux.
Nous avons choisi de changer les noms des signaux pour que le code soit plus lisible et plus parlant, les conventions prises sont toutes détaillées dans le github de notre projet.
Notre processeur ne possède pas encore de bypass qui sont en cours de création, mais il est capable de compiler du C ou de l'assembleur.
L'implémentation RISC-V que nous avons surnommé v1.0 est à ce jour totalement finalisé. Nous avons en effet réalisé multiple tests C tel que la suite de Fibonacci récursive ou encore un algorithme de calcul de PGCD.

3.1 Résumé graphique

Tous les membres de notre projet ont participé à l'ensemble de la conception, mais certains membres se sont plus concentré sur certaines taches. Ainsi, le tableau ci-dessous indique les taches accompli et celles qui sont en cours, et qui a majoritairement contribué à la réalisation de la tache.

	Janvier	Février	Mars	Avril	Mai			
Documentation RiscV	X							
Conception CORE RiscV v1.0								
Etage IFETCH		X						
Etage DECODE		X						
Etage EXEC		X						
Etage MEMORY		X						
Etage WRITEBACK		X						
Débogage CORE		X						
Nettoyage du programme			X					
Conceptions Caches								
Mis à jour Mips3000R		X						
Documentations Mips3000R		X						

X : Représente le mois où la tache a été finalisée.

Timothée Le Berre	
Louis Geoffroy Pitailler	
Kevin Lastra	

3.2 Validation

Pour validé notre implémentation nous avons fonctionné comme suit :
- Dans un premier temps nous avons réalisé des test benchs pour chacun des étages afin de vérifié leur fonctionnement. Ces test consistait à envoyé des signaux avec des valeurs aléatoire dans l'étage testé et de regardé ce que nous obtenions en sorie,
- Dans un second temps nous avons compilé quelque programme assembleur assez simple avec une ou deux instructions, nous avons ensuite complexifié les programmes et nous avons conçu des tests comportants toutes les instructions d'un meme type. Nous avons par exemple un test qui s'appelle test_all_ops.squi test toutes les instructions de type I,
- Enfin nous avons écrits des programmes C comme la suite de fibonacci ou un algorithme de PGCD que nous avons compilé et nous avons vérifié que tout fonctionnait correctment.

4 Problèmes rencontrés lors de la conception

Plusieurs problématiques ont surgi depuis le début du projet mais grâce au travail d'équipe et a la bonne répartition des taches, nous avons pu passer outre.

La première difficulté majeure a été celle de rendre fonctionnelle le code systemC [2] du MIPS qui nous a été fourni, en effet les conventions utilisées étaient difficiles à comprendre et le code était peu documenté. Une fois le code rendu compilable, il a fallu le décortiquer complètement ce qui a pris beaucoup de temps.
La deuxième difficulté rencontrée aura été la synthèse de la spécification RISCV, il a en effet fallu faire le trie entre ce que nous comptions implémenter ou pas et il nous a fallu bien comprendre le jeu d'instruction afin de l'implémenté correctement dans DECOD.
Enfin le débogage du Core complet pour compiler des programmes C nous aura pris du temps, en effet nos tests assembleur se sont vite monté insuffisant et il nous a fallu trouver ce que ne fonctionnait pas lors de la compilation de nos programmes C.
Pour finir une difficulté que nous rencontrons actuellement, mais qui devrait bientôt être résolu est la mise en place des bypass, en effet nous avions mis en place tout un système d'invalidation dans le banc de registre dans le but de geler le pipeline en cas de dépendance de données, mais nous nous sommes aperçu que cette invalidation n'était plus nécessaire une fois les bypass correctement implémenté.

5 Objectifs restants

Le tableau ci-dessous fait office de Roadmap, il s'agit d'un récapitulatif graphique des objectifs que nous nous sommes fixés et les deadlines correspondantes.

	Fin Février	Fin Mars	Fin Avril	Fin Mai
Débogage & implémentation des bypass		X		
Implémentation de la partie Kernel				X
Implémentation des caches		X		
Implémentation SS2 & débogage complet				X

Les Bypass devraient donc être bientôt terminé, il nous faudra ensuite décortiquer tout le cours de Mr. Franck Wajsburst qui nous a été fourni, afin d'implémenter la partie Kernel.
Nous n'avons pas encore défini qui ferait quoi pour la partie kernel et le SS2 car nous n'avons pas vraiment d'idée de ce qu'il faut faire pour le moment donc il est difficile de répartir les taches.En effet, nous n'avons jamais implémenté de partie Kernel, nous allons donc tâtonner jusqu'à comprendre comment cela fonctionne et avoir une idée de comment implémlenter ca pour enuite répartir les taches. Cela veut donc dire que chacun de nous va chercher à comprendre comment implémenter la partie Kernel.
Enfin s'il nous reste du temps nous passerons le processeur en Super-scalaire, mais il nous est assez difficile de nous projeter jusque-là étant donné que nous n'avons pas de réelle idée de quels vont être les difficultés d'implémentation de la partie Kernel.

References

[1] https://en.wikipedia.org/wiki/IBM_System/370

[2] https://www.accellera.org/

[3] https://en.wikipedia.org/wiki/IBM_document_processorsIBM_801

[4] Waterman, A., Lee, Y., Patterson, D., Asanovic, K., level Isa, V. I. U. (2014). The RISC-V instruction set manual. Volume I: User-Level ISA', version, 2.

[5] Waterman, A., Lee, Y., Avizienis, R., Patterson, D. A., Asanovic, K. (2015). The risc-v instruction set manual volume 2: Privileged architecture version 1.7. University of California at Berkeley Berkeley United States.

[6] Asanović, K., Patterson, D. A. (2014). Instruction sets should be free: The case for risc-v. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146.

[7] Utting, M., Kearney, P. (1992). Pipeline specification of a MIPS R3000 CPU. Technical Report 92-6, Software Verification Research Centre, Department of Computer Science, University of Queensland.

[8] David A. Patterson John L. Hennessy (2021). Computer organization and design RISC-V edition, second edition.

[9] Jurij Šilc, Jurij Silc, Borut Robic, Theo Ungerer (1999). Processor architecture : From dataflow to superscalar and beyond.