Processeur MIPS

Départements : TIC

Unité d'enseignement ARO

Auteurs : **Bastian Chollet  
Kevin Ferati**

Professeur : **Marina Zapater**

Assistant : **Mike Meury**

Classe : **C**

Salle de labo : **A09**

Date : **01 mai 2022**

# Introduction

Dans le cadre du cours d’ARO à l’HEIG, nous devons concevoir un processeur à jeu d’instructions réduit (RISC) simplifié. Les instructions supportées seront celles du kit d’instruction ARM THUMB (16b).  
  
Les composants développés sont :

1. FETCH
2. DECODE
3. EXECUTE & MEMORY ACCESS

Pour cela, nous avons travaillé sur une machine virtuelle Ubuntu avec l’outil Git. En général, l’un de nous travaillait pendant que l’autre avançait le rapport.

# FETCH

## Introduction

Dans un processeur, le composant FETCH est celui qui va déterminer, en fonction du Program Counter (un registre sauvegardant l’instruction à traiter) et de l’instruction de saut, l’adresse de l’instruction qui doit être exécutée. Plus précisément, la sortie du FETCH est une adresse transmise à la mémoire d’instructions, laquelle restitue l’instruction correspondante.

La mémoire d’instruction est une Read-Only Memory qui garde les instructions d’un programme.

FETCH prend en entrée (principalement) :

* L’instruction en cours afin de déterminer s’il doit y avoir un saut (conditionnel, inconditionnel, …). S’il n’y a pas de saut, la sortie sera le PC incrémenté par le nombre d’octets dans une instruction (soit, 2) ;
* L’adresse contenue dans le Program Counter (afin de pouvoir l’incrémenter si nécessaire) ;
* L’adresse de saut s’il y a un saut ;
* Le bus de contrôle qui indique si une instruction est un saut et est conditionnelle ainsi que la condition à tester. Ceci est normalement fait dans le composant DECODE, mais pour des questions d’essais nous devons l’implémenter déjà ici.

FETCH renverra :

* L’adresse pour la prochaine instruction ;
* Les données de l’instruction courante ;
* La valeur future du Program Counter.

## Analyse et conception

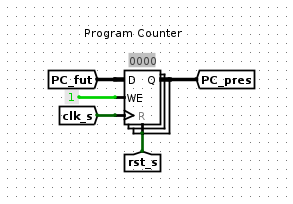
Le Program Counter (PC) doit se situer normalement dans une banque de registre. À notre stade, ayant besoin uniquement de ce registre, nous l’implémentons à part. De là, nous pourrons tester des simples instructions pour voir si le PC a été correctement implémenté.  
  
Pour la gestion du saut, normalement la détermination du saut (c’est-à-dire s’il y en a un et s’il est conditionnel) se fait dans la partie EXECUTE. Vu qu’on en besoin, on implémente un mini-déterminateur. Ceci se fait depuis les données de l’instruction. Ce déterminateur va analyser l’instruction et comparer les 4 premiers bits pour savoir le type de saut.

La gestion des sauts conditionnels (c’est-à-dire le calcul des conditions et l’enregistrement des résultats de la comparaison dans un registre) se faisant dans l’EXECUTE, nous testons les sauts conditionnels en mettant « à la main » en spécifiant les constantes de résultats de comparaison à la main. Par exemple, dans le cas d’un BEQ, il faut que le bit à la position 0 (le LSB) soit à 1. C’est la situation que nous allons prendre ce qui permettrait de faciliter les tests.

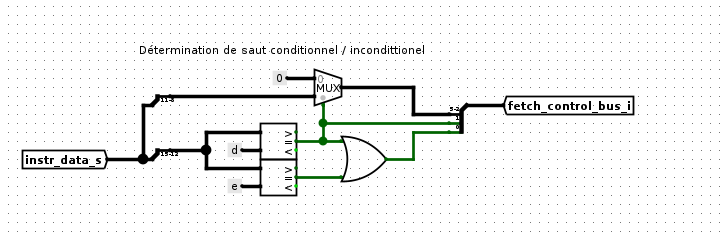
## Réalisation

Nous n’avons eu aucun problème à réaliser les différentes parties du composant.

Résultat du Program Counter : Ici, rien de spécial.

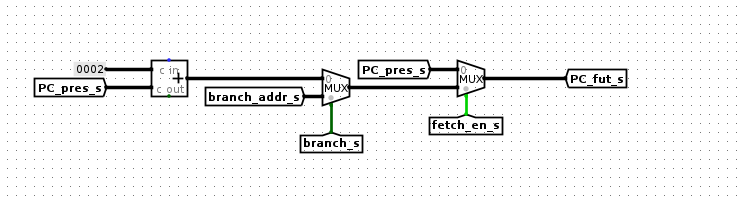


Détermination du saut. Ici, rien de difficile : on vérifie les premiers bits pour savoir si c’est un saut et on récupère les conditions si c’est le cas.



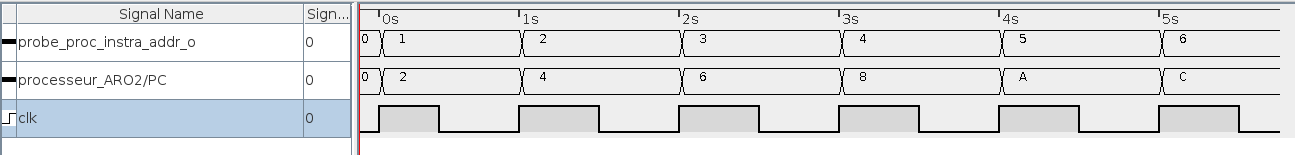
Pour le calcul de l’adresse de saut, nous avons respecté le manuel et les cours : Une multiplication par 2, suivi d’une extension à 16b et d’une addition de 4 et du PC.

Sélection de l’adresse d’instruction finale : S’il y a un saut, le futur PC est l’adresse de saut. Sinon, ce sera l’adresse courante incrémentée de 2. Dans tous les cas, si le FETCH est désactivé, l’adresse ne change pas.



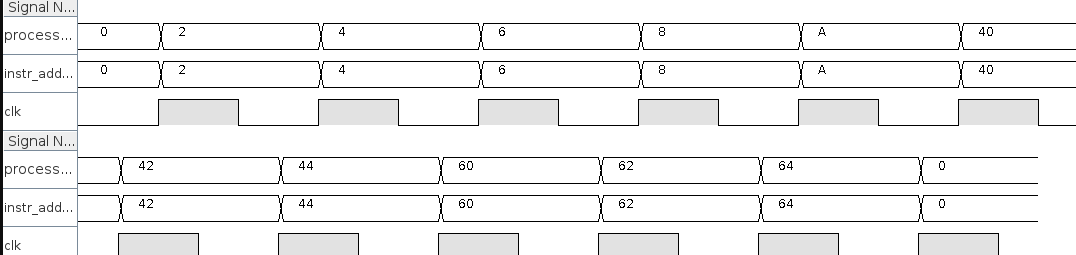
## Test

Comme première simulation, nous avons essayé, après implémentation du PC, son incrémentation. On peut voir le résultat ci-dessous. La première ligne correspond à l’adresse de sortie du FETCH (sur 15 bits) et la deuxième au PC. On observe qu’à chaque horloge, le PC est incrémenté de deux et l’adresse de 1, ce qui est correct. Il ne faut pas oublier que pour cette dernière les bits 15-1 seulement sont prises et on ignore le dernier bit.



Une image contenant texte

Description générée automatiquementIl faut ensuite simuler des sauts. Pour cela, nous avons préparé un programme assembleur avec deux sauts inconditionnels et un conditionnel. Le programme commençant à 0x0, après avoir traité à l’instruction 0xA le programme devrait vers un saut sur 0x40. Après l’instruction à 0x44, le programme saute à 0x60. Ensuite, à 0x64, il revient à 0x0.

Résultat :  
  


La première ligne est le PC et la deuxième est l’adresse d’instruction (cette fois à 16b pour plus de clarté). Nous observons que ces deux éléments suivent le chemin précédemment énoncé : il y a 3 sauts (A => 40, 44 => 60 et 64 => 0). Ainsi, le FETCH semble fonctionner.

## Feedback

Nous avons reçu un simple retour, le reste fonctionnant très bien. En effet, nous avions utilisé des extenseurs qui complexifiait les schémas. Nous les avons simplement remplacés par des splitters.

# DECODE & BANK REGISTERS

## Introduction

La banque de registre (BR) est un composant regroupant les registres disponibles pour lecture et écriture. C’est le composant mémoire le plus proche du CPU et donc le plus rapide mais aussi le plus coûteux.

Dans notre cas, il est composé de 8 registres de 16b. La notation est R<numéro de registre> à partir de 0.

* Les registres 0 à 4 sont n’ont pas de fonctions particulières autre que le stockage.
* Le registre 5 est nommé le Stack Pointer et contient l’adresse vers l’instruction de la stack courante.
* Le registre 6 est nommé le Link Register et contient l’adresse de retour vers l’instruction à exécuter à la fin d’une interruption. Sa gestion se fait au moyen du composant « LR\_Manager », déjà fourni.
  + LR\_Manager indique les données à écrire dans le LR et si l’écriture est activée.
  + L’écriture dans le LR est activée s’il y a un lien, si l’écriture est activée « manuellement » avec sélection du registre 6.
* Le registre 7 est le Program Counter, mentionné lors du FETCH. La possibilité d’écrire dedans dépend de l’existence d’un BL.

Les entrées principales du composant sont :

* Trois entrées de sélection de registre pour lecture, tous en 3b (car 8 registres). Le contenu des registres sélectionnés par deux de ces sélecteurs (M, N) ira dans le composant EXECUTE, et le dernier (MEM) servira dans les cas de calcul d’adresse pour écrire en mémoire ;
* Une entrée de sélection de registre pour écriture (D), sur 3b ;
* Les données à écrire dans ledit registre ;
* Le bus de contrôle sur 2b (le LSB n’est pas utilisé et le MSB active ou non l’écriture) ;
* Une entrée spéciale pour le PC qui va définir ses données.

Le composant DECODE fait, comme son nom l’indique, un décodage de l’instruction transmise par FETCH. Plus précisément, ce composant :

* Sépare une instruction par ses différentes opérandes et valeurs immédiates ;
* Sélectionne l’instruction à exécuter. Ceci se fait au moyen du composant Main Control Unit.
  + Ce composant compare simplement les premiers bits d’une instruction et active le flag correspond à l’opération. Il gère aussi les contrôles bus de différents composants.

## Analyse et conception

### Banque de registre

Pour l’écriture, il nous faut activer le registre sélectionné par l’opérande d’écriture (reg\_d\_sel\_i). Ceci peut se faire au moyen d’un décodeur. L’écriture est activée seulement si le MSB du bus de contrôle est activé et que la banque est activée.

Pour la lecture, un simple MUX de 8 choix par sélecteur de registre en lecture, donc 3 MUX suffit.

### Decode

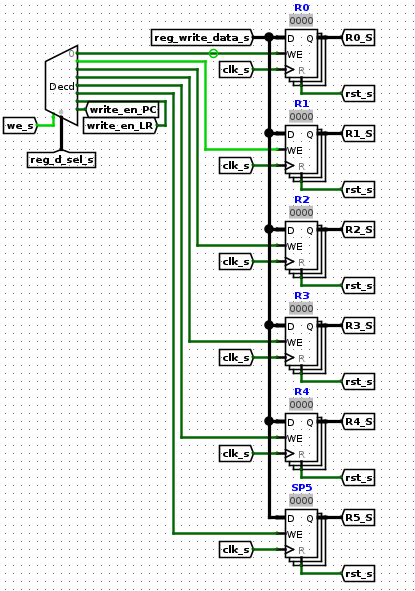
Nous distinguons plusieurs cas :

* Sélection des registres ;
  + Pour ceux-là (sélecteur N, M) un MUX suffit amplement car il faut sélectionner différentes valeurs en fonction d’un sélecteur.
* Ajout de décodage de certaines instructions ;
  + Pour cela, dans le composant opcode\_supported\_unit, il suffit d’ajouter les comparaisons avec les premiers bits des instructions manquantes.
* Les bus de contrôles des différents composants. Pour ceci, il suffit de mettre ne porte OU pour les différentes sorties des bus.

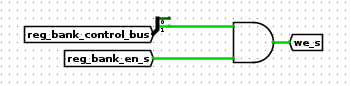
## Réalisation

### Banque de registre

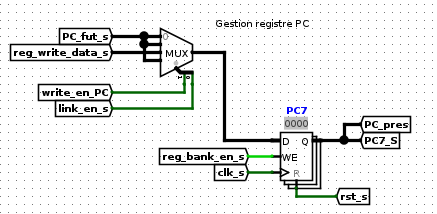
Pour l’écriture, nous avons, comme convenu, implémenté un simple décodeur qui active le registre correspond au sélecteur « D » : avec le sélecteur D à 001 (1) :



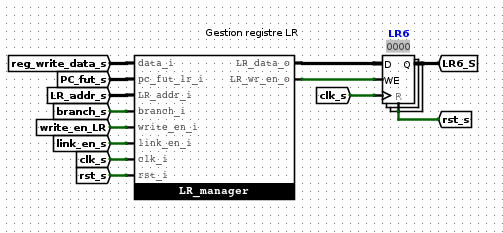
« we\_s » (signal qui active le décodeur et donc l’écriture) correspond à :

  
Il faut donc que la banque ET que le MSB du bus de contrôle soit activés pour que l’écriture le soit.

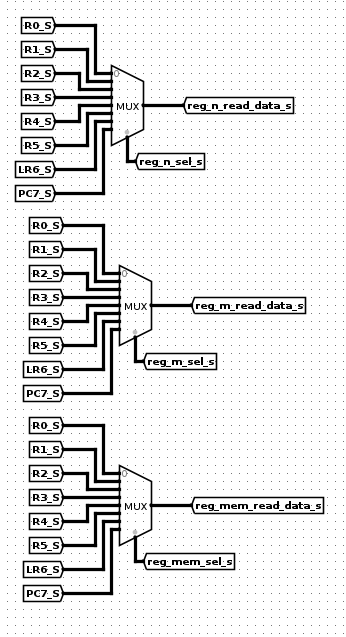
Pour respecter le cahier des charges concernant l’écriture dans le registre PC, nous avons créé un MUX avec 2b de sélection :



Pour le LR, nous avons correctement implémenté le LR Manager :



Pour la lecture, comme convenu durant l’analyse, 3 MUX avec les 3 différents sélecteurs ont suffi :



### DECODE

Cette partie a surtout consisté en une série de tirage de câble. Tout d’abord, il s’agissait de créer les sélecteurs pour N, M, D. Pour cela, nous avons simplement suivi le cahier des charges en implémentant MUX sur MUX :

## Test

### Banque de registres

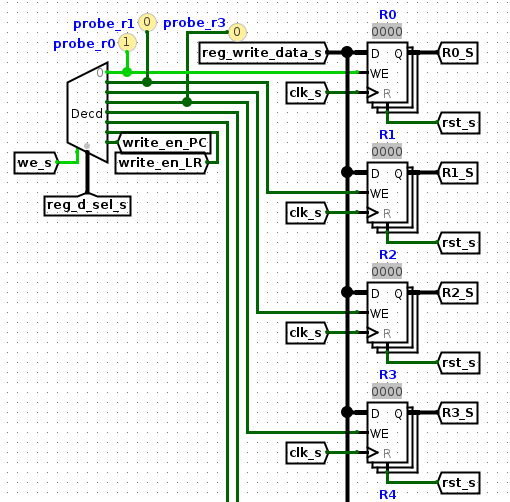
Programme de test :

Une image contenant texte

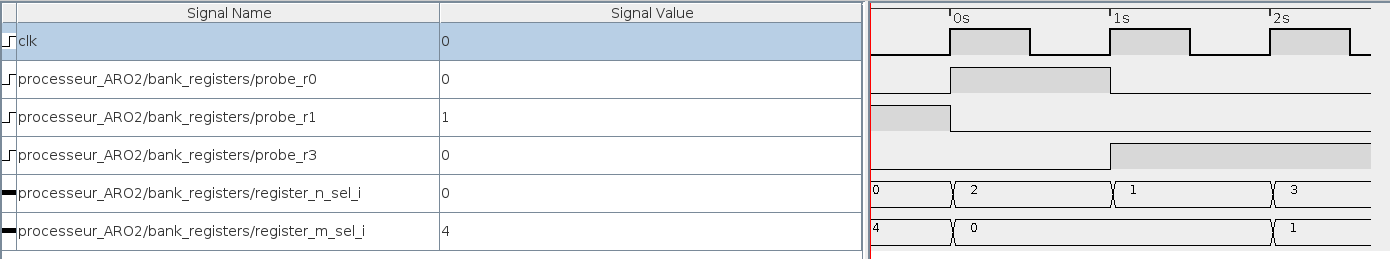
Description générée automatiquement

Pour la lecture, nous observons les sélecteurs.

Pour l’écriture, nous observons la sortie du décodeur d’activation d’écriture de R0, R1 et R3 :



Résultat :



|  |  |  |
| --- | --- | --- |
| Instruction | Comportement voulu | Comportement |
| MOV r1, #5 | R1 activé pour écriture | Probe\_r1 = 1 |
| MOV r0, #21 | R0 activé pour écriture | Probe\_r0 = 0 |
| ADD r3, r1, r0 | R3 activé pour écriture  R1, r0 sélectionné | Probe\_r3 vaut 1  N\_sel\_i = 1 (registre 1) et m\_sel\_i = 0 (registre 0) (OK) |
| AND r3, r1 | R3 activé pour écriture et lecture, r1 pour lecture | Probe\_r3 = 1, n\_sel\_i = 3, m\_sel\_i = 1 (OK) |

# EXECUTE & MEMORY ACCESS

## Introduction

## Analyse et conception

## Réalisation

## Simulation

## Test

## Feedback

Date : Date rendu du rapport

Noms des étudiants : Etudiant A Etudiant B

# Conclusion

Qu’a-t-on tiré du projet,quels problèmes rencontrés (organisationnels, méthodologiques, techniques, …) comment refaire certain point et remarques personnels, utilité des PDF des cours

# Annexe

Documents annexés avec numérotation et commentaires.