RAPPORT
ADO

2022-2023

ANDY TORRES
STEPHANE RAMAHEFARINAIVO
585k

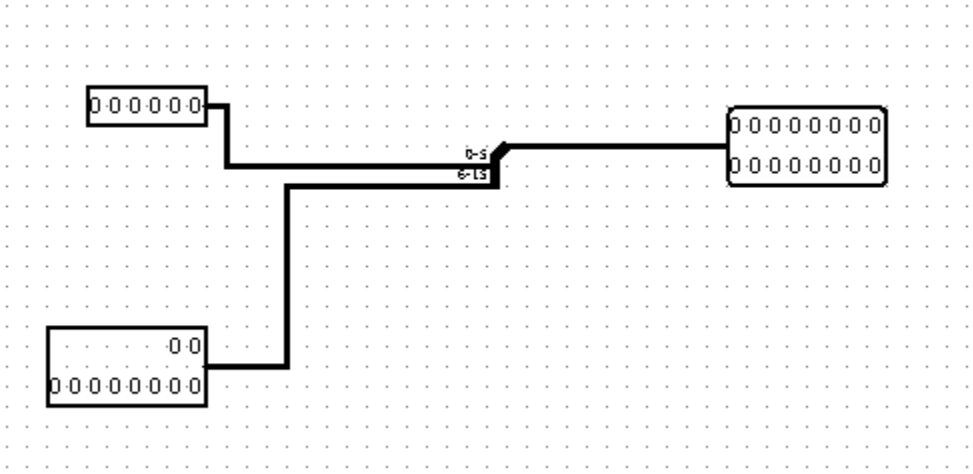
Kathleen16

But du projet :
Construire un mini processeur de 16 bits Kathleen 16.

Pour ce faire 4 circuits sont important dans le fonctionnement de ce processeur :

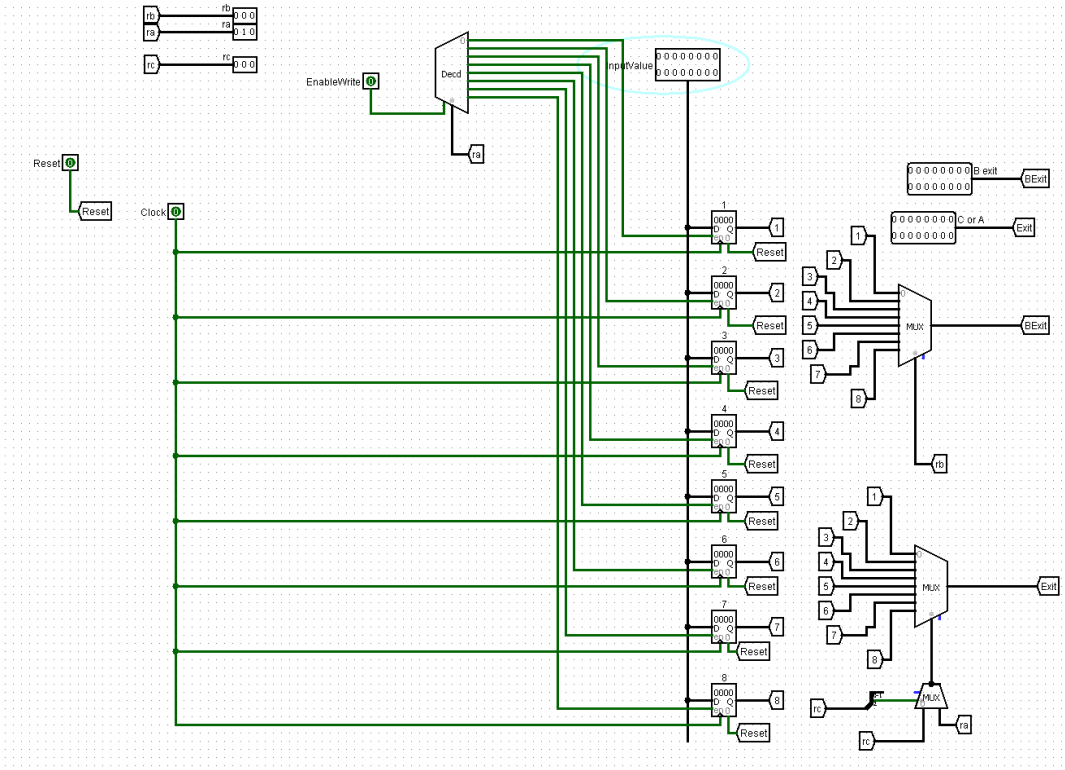
- Extenseur 16
- Banc de registre
- UAL
- IFU

Extenseur 16 :



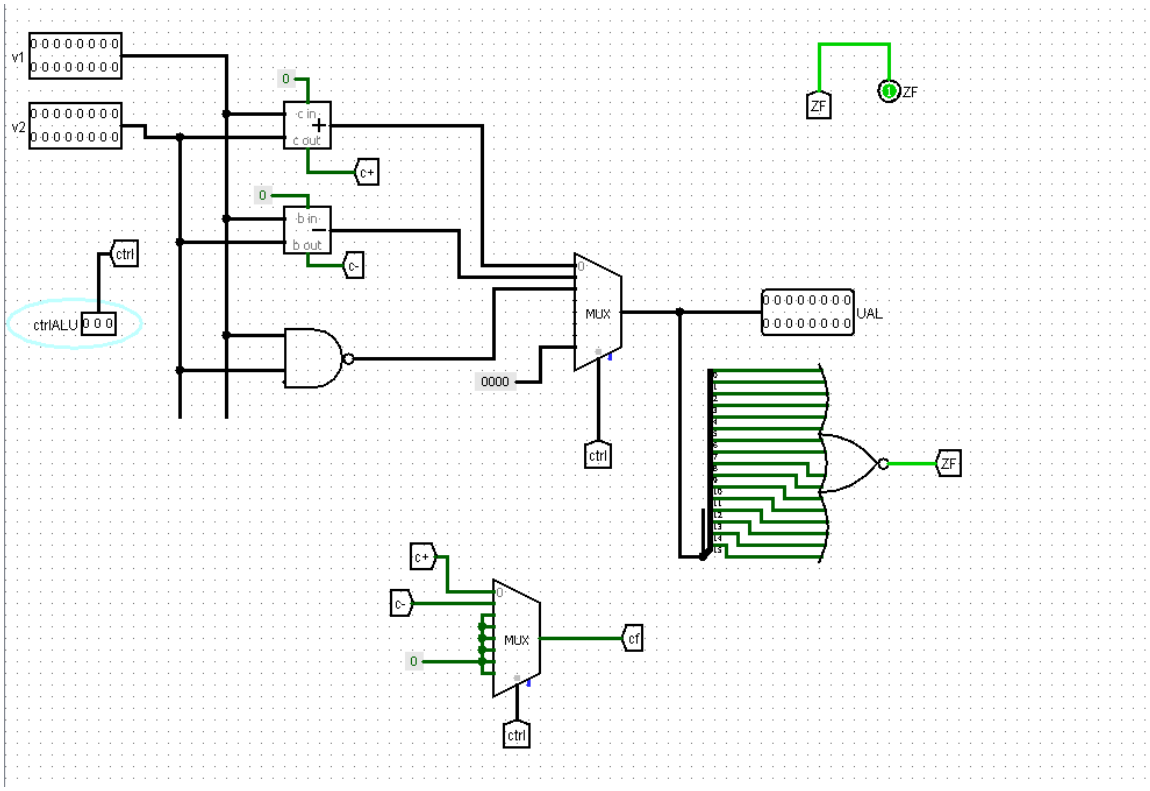
Un circuit utile pour l'instruction lui.
Le circuit prend en paramètre une valeur sur 10 bits qu'on va ensuite placer dans un splitter qui est relié à une "constante" de 6 bits qui feront office de décalage de 6 pour obtenir une variable sur 16 bits.

Banc de registre :



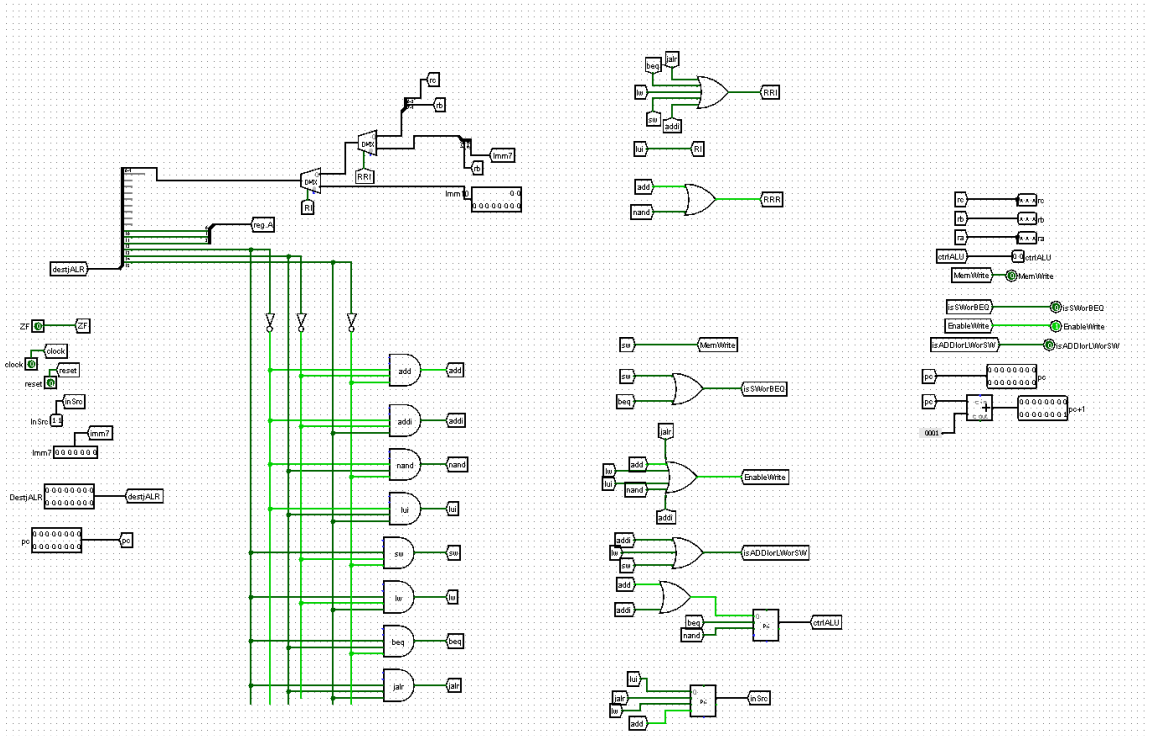
Le banc de registre est un circuit a la différence de son n'ayant pas pour rôle de stocker des valeurs à l'instar d'une RAM, plutôt de pouvoir la placer temporairement dans un registre pour mieux les manipuler.
Prenant une valeur de 16 bits, un bit EnableWrite, et 3 paramètres rc, rb et ra. EnableWrite peut être définie comme un signal permettant de dire au circuit d'écrire dans un registre la valeur de 16 bits passée. Ra contient 3 bits donnant l'information dans quel registre placer temporairement la valeur, rb accompagné d'un multiplexeur va alors en fonction de ses bits choisir quel registre prendre pour stocker, prendre la valeur stockée et la placer dans la sortie. On fait de même avec rc et ra.
On utilise aussi une clock et un reset pour pouvoir faire fonctionner le circuit ou le réinitialiser en cas de besoin.

UAL :

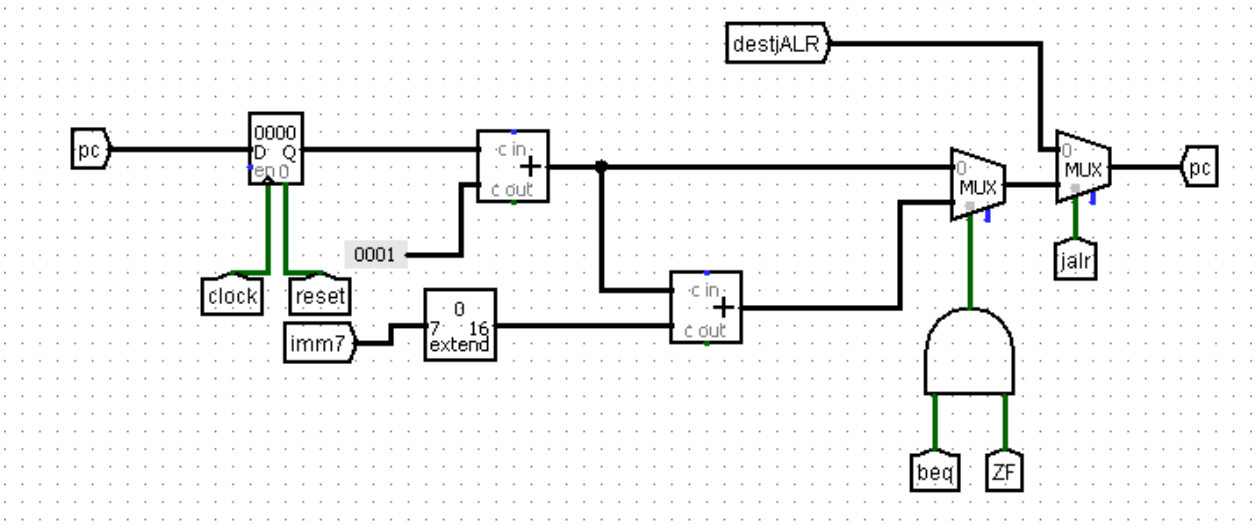


L'UAL est un circuit permettant de réaliser trois calculs différents à partir de ses 2 paramètres d'entrées (v1 et v2 sur 16 bits) en fonction du signal de contrôle (ctrl) afin de générer les drapeaux CF(carry flag) et ZF(zero flag) en plus du résultat de l'opération souhaitée stocké sur 16 bits. Ces drapeaux sont générés de la manière suivante: ZF est égal à 1 lorsque tous les bits du résultat sont 0, d'où le choix d'une porte NON-OU sur l'ensemble des bits du résultat. CF est quant à lui égal à 1 uniquement lorsqu'il y a une retenue sur le bit de poids fort d'où l'importance du calcul et de la conservation des carry des opérations via les variables c+ pour l'addition et c- pour la soustraction.

IFU:



IFU (Instruction Fetch Unit) est le plus gros circuit du processeur Kathleen, en effet son rôle est le plus conséquent, c'est le premier endroit où les instructions sont "analysées" et envoyées directement au processeur pour les exécuter.
Grossièrement, il prend une adresse/instruction de 16 bits et va la décomposer pour obtenir différents éléments pour comprendre et l'exécuter.
Comme dit précédemment, IFU prend une valeur sur 16 bits, l'adresse de l'instruction, avec un splitter on décompose, les 3 premiers bits font référence à l'Opcode qui fait lui référence à un format et instruction MIPS (add, addi…).
Il existe 3 formats en RRI, RRR et RI, chaque format possède une ossature de l'adresse de 16 bits différentes.
Les 6 premiers bits sont très similaires pour tout format : Opcode → Reg A, avec son Opcode on connaît dans quel format l'instruction se trouve. Avec cela, on peut déjà (tout dépend de l'instruction) faire des actions, par exemple si l'Opcode donne l'instruction add, alors on sait qu'on va devoir écrire donc on informe en plaçant un bit à 1 (EnableWrite) dans une variable qu'il va falloir écrire dans un registre. De plus, on sait que add est une instruction de RRR alors on stocke également cette information.
Avec le format, on va pouvoir manipuler les 10 bits restants, en utilisant des multiplexeur qui rempliraient des variables, si RI est utilisé alors les 10 bits seront alors stockés dans Imm 10, donnant ainsi Opcode → Reg A → Imm 10.



Cette partie du circuit permet de manipuler PC (Program Counter) l'adresse de l'instruction courante et suivante.
En fonction de l'instruction utilisée, ce circuit va soit donner l'adresse suivante soit garder son adresse initiale. On ajoute au pc + 1 pour pouvoir sauter à la prochaine instruction, si BEQ ou ZF sont utilisées, alors on va devoir faire PC + 1 - 10 permettant de revenir en arrière aux instructions précédentes d'où l'addition avec imm7 sous extendeur de 16 bits. Avec cette adresse, on va choisir avec un multiplexeur, si l'instruction Jalr est ordonnée alors on saute à la prochaine instruction qu'on a calculée précédemment, si Jalr n'est pas utilisé alors PC+1 est inutile car aucun saut n'est demandé.

Partie 2:

1.

Voici la traduction du code Kathleen16 en langage d'assemblage:

- addi $r1, $r0, 0 → 001 001 000 0000000
  L'instruction "addi" est dans le format RRI (3 bits pour l'opcode et les registres A et B ainsi que 7 bits pour la valeur immédiate. Ainsi, on a 001 pour l'opcode, 001 pour le registre r1, 000 pour le registre r0 et 0000000 pour coder la valeur 0.

- addi $r2, $r0, 0 → 001 010 000 0000000
  On effectue la transcription de manière analogue. A noter que le registre r2 est codé par la valeur 2 soit 010 tout comme r1.

- addi $r3, $r0, 60 → 001 011 000 0111100
  Ici, r3 est codé par la valeur 3 soit 011 et 60 par sa valeur binaire sur 7 bits.

- beq $r2, $r3, endfor → 110 010 011 0000011
  L'instruction "beq" est également dans le format RRI donc on prend l'opcode de "beq" soit 110 et endfor est codé par le nombre d'instructions que l'on doit sauter pour arriver à l'étiquette "endfor" soit 3 sur 7 bits.

- add $r1, $r1, $r2 → 000 001 001 010
  L'instruction "add" est dans le format RRR (3 bits pour l'opcode et les registres A,B et C). Ainsi, chaque registre est codé comme précédemment par la valeur de son numéro sur 3 bits et l'opcode vaut 000 pour cette instruction.

- addi $r2, $r2, 3 → 001 010 010 0000011
  Cas analogue à la première instruction.

- beq $r0, $r0, for → 110 000 000 1111101
  Cas analogue à la première instruction "beq", cependant, "for" est codé par le nombre d'instruction sauté depuis l'étiquette "for" * (-1) soit -3. Ainsi, il suffit de prendre le complément à 2 de 3 sur 7 bits.

- beq $r0, $r0, endfor → 110 000 000 1111111
  Cas analogue au précédent mais cette fois-ci, il n'y a d'une instruction séparant celle-ci à l'étiquette "endfor" donc il suffit de calculer le complément à 2 de 1 sur 7 bits.

  1.

  L'instruction "li" peut s'implémenter de la manière suivante :

- addi $r1, v, 0
- lui $r2, v
- add $r1, $r2, $r1

En effet, on commence par stocker les 7 bits de v dans le registre r1 et initialiser les 9 bits restants à 0(xxxxxxx000000000), puis on utilise l'instruction "lui" pour stocker les bits manquants dans r2 (0000000xxxxxxxxx). Finalement, on ajoute le registre r2 à r1, ce qui permet de récupérer les 16 bits sans perdre d'information étant donné que 0 est l'élément neutre de l'addition.

2.

- not $ra, $rb → nand $ra, $rb, $rb
  On utilise la propriété suivante: $\neg(A \wedge A) \longleftrightarrow \neg A$

- and $ra, $rb, $rc → nand $ra, $rb, $rc puis not $ra, $ra
  On utilise la propriété suivante: $\neg(\neg(A \wedge B)) \longleftrightarrow A \wedge B$

- or $ra, $rb, $rc → not $rb, $ra ; not $rc, $rb ; and $ra, $rb, $rc ; not $ra, $ra
  On utilise la propriété suivante: $\neg(\neg A \wedge \neg B) \longleftrightarrow A \vee B$

- sub $ra, $rb, $rc → not $rc, $rc ; addi $rc, $rc, 1 ; addi $ra, $rb, $rc
  On utilise la propriété suivante: $A + (\neg B) \longleftrightarrow A - B$ en prenant le complément à 2 de B

- xor $ra, $rb, $rc → or $r1, $rb, $rc ; nand $r2, $rb, $rc ; and $ra, $r1, $r2
  On utilise la propriété suivante: $(A \vee B) \wedge \neg(A \wedge B) \longleftrightarrow A \oplus B$

- nop → beq $r0, $r0, 0
  Cette instruction ne fait rien, c'est pourquoi le test "$r0 == $r0" retournera vrai et on effectuera un saut sans effet à PC+1 comme Imm7 vaut 0 dans ce cas.

Conclusion :

Un projet assez mitigé dans l'ensemble, la difficulté variait entre les circuits surtout lors de la fabrication de l'IFU qui était vraiment pas facile, un Kathleen 16 totalement fait mais on est conscient qu'il est non-fonctionnel, on manquait de temps pour pouvoir peaufiner et totalement trouvé où le circuit bloquait malgré que pris individuellement chaque circuits fonctionnent plutôt bien.