Rapport de Projet

Rédigé par Raphaël Trancoso & Cédric Laguerre

Université Paris-Diderot - 9 janvier 2017

Projet Circuits Et Architecture

TABLE DES MATIÈRES

Introduction

Dans le cadre du module circuits et architecture de la première année de Master à l'Université Paris-Diderot, nous avons construit un processeur permettant d'implanter certaines instructions du LC-3. C'est le sujet de ce document.

Contacts

Pour toutes informations relatives à ce projet vous pouvez nous contacter aux adresses mails suivantes :

- Raphaël Trancoso : raphael.trancoso@laposte.net
- Cédric Laguerre : ced.laguerre@gmail.com

Démarche suivie

Pour mener à bien ce projet, nous avons suivi la démarche suivante :

- Comprendre le fonctionnement général du micro-processeur LC-3 et création des diagrammes temporels.
- Tester les simulations des fichiers mémoires « .mem » des différentes opérations arithmétiques.
- Implémentations des circuits.

Le processeur

Spécifications du processeur

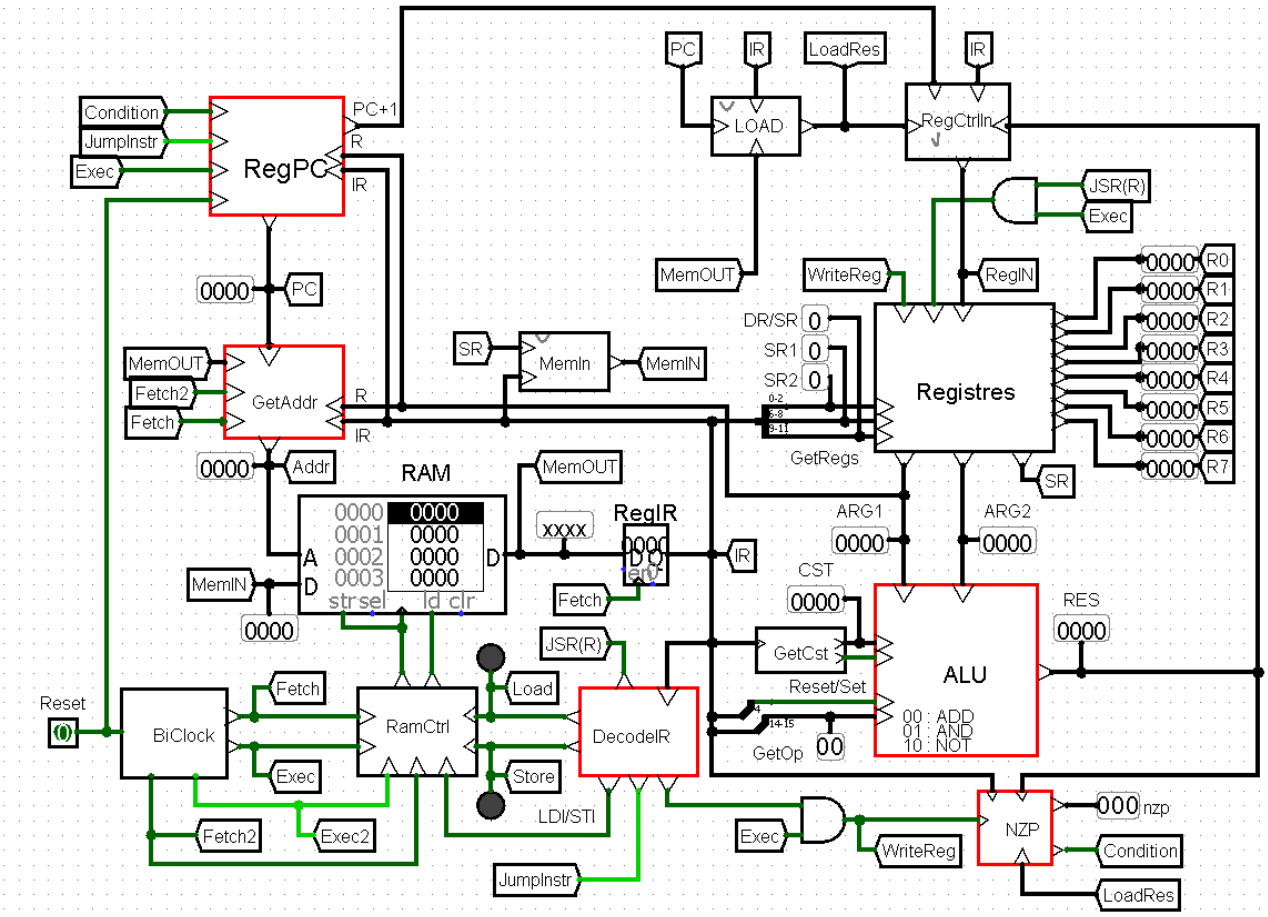
Le micro-processeur LC-3 est composé de :

- un banc de 8 registres généraux de 16 bits appelés R0,R1,…,R7.
- Une unité de calcul (ALU : Unité Arithmétique et Logique)
- Une unité d'adressage
- Une unité de contrôle (DecodeIR)

Le processeur LC-3 travaille sur des mots de 16 bits. Il a deux bus de communication avec deux mémoires externes qui sont :

- La mémoire de données
- La mémoire d'instruction

Organisation du processeur

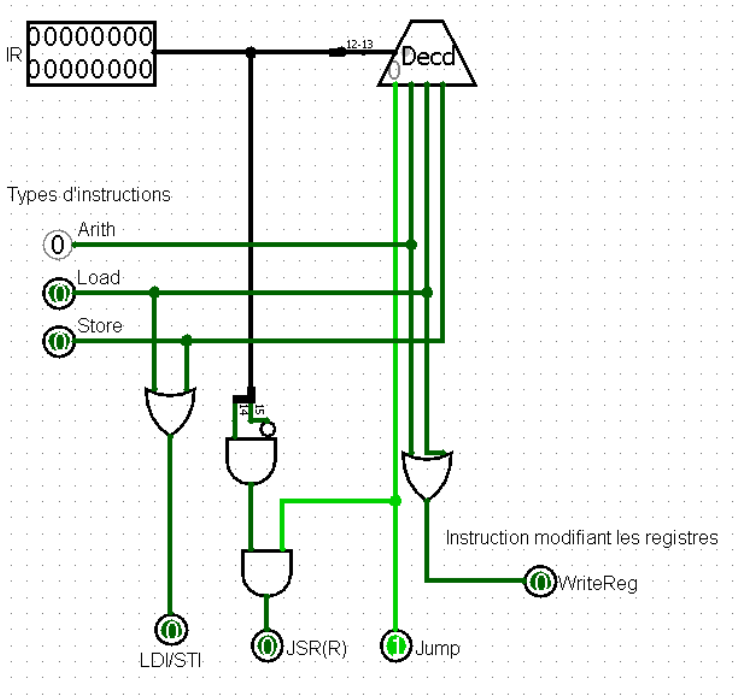


Jeu d'instruction du processeur

MSB	LSB			
	0 0	0 1	1 0	1 1
0 0	BR	ADD	LD	ST
0 1	JSR(R)	AND	LDR	STR
1 0	RTI	NOT	LDI	STI
1 1	JMP	SET/ RESET	LEA	TRAP

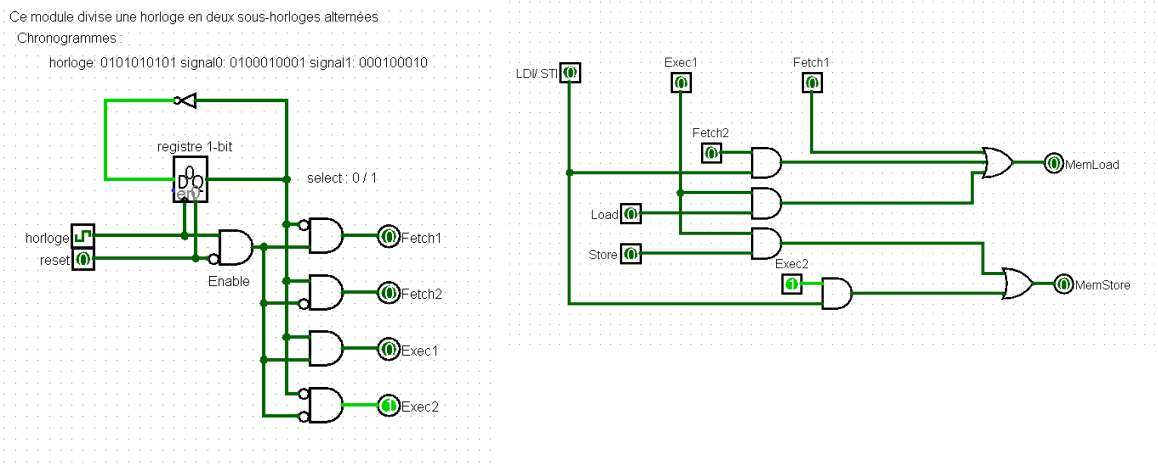
Réalisation - améliorations apportées

DecodeIR



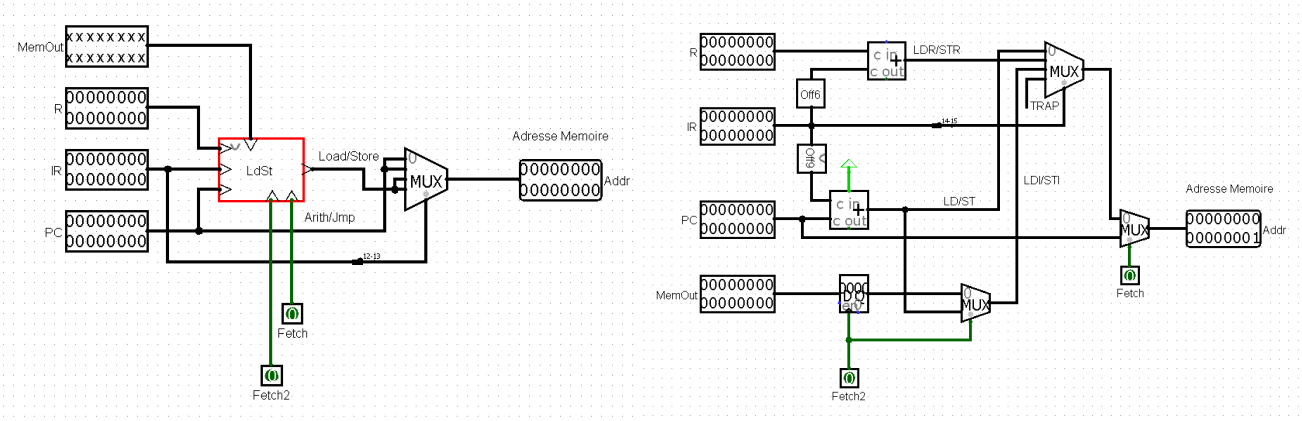
Elle permet de décoder l'instruction, c'est à dire qui interprète les signaux du registre d'instruction et ensuite renvoie le type d'instruction ainsi que la permission de modifier le registre selon l'instruction.

BiClock/RamCtrl



Pour pouvoir gérer les instruction LDI/STI nous avons du diviser les 2 signaux en 4 signaux, pour faire les deux accès mémoires en fonction de l'horloge.

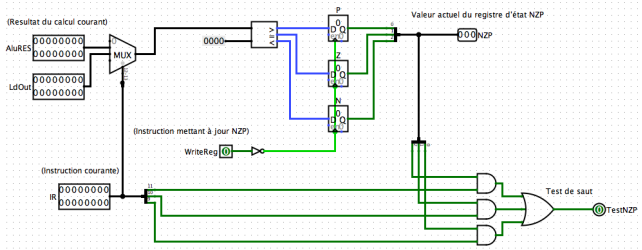
GetAddr



Cela retourne une adresse en fonction de l'instruction :

- Si l'instruction est un chargement ou un rangement, alors l'adresse retournée sera le contenu du registre BaseR codé sur 3 bits [8,6] additionné avec un Offset codé sur 6 bits [5,0] codé pour l'instruction LDR/STR et pour une instruction LD/ST, l'adresse retournée sera le compteur PC additionné par un Offset de 9 bits [8,0] codé dans l'instruction.
- Enfin pour l'instruction LDI/STI, elles calculent, comme les instructions LD et ST, la somme du compteur de programme et de l'offset. Elles chargent la valeur contenue à cette adresse (MemOut) puis utilisent cette valeur à nouveau comme adresse pour charger ou ranger le registre.

NZP

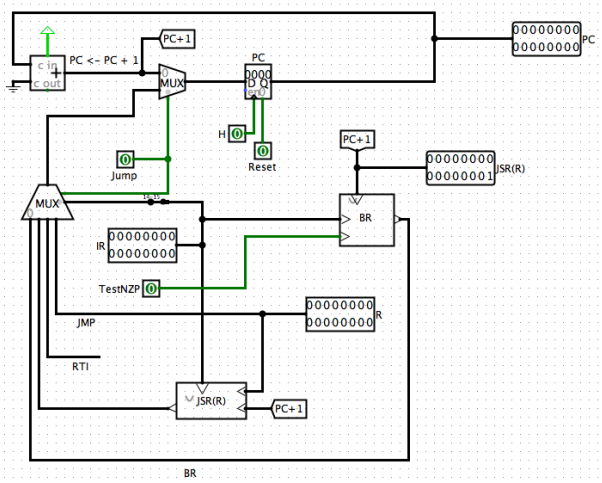



```
.ORIG x3000

        AND R0,R0,0
        ADD R1,R0,5
        ADD R0,R0,9
        AND R0,R0,R1

.END
```

Le NZP sont des indicateurs testés par les branchements conditionnels (teste si la valeur mis en entrée est négative,

nulle ou positive). Les 16 bits entrants sont le résultat de l'ALU ou un chargement (LEA/LD/LDR/LDI). Si le codeOP de l'instruction est un branchement, il envoie le signal de testNZP pour le branchement conditionnel.

RegPC



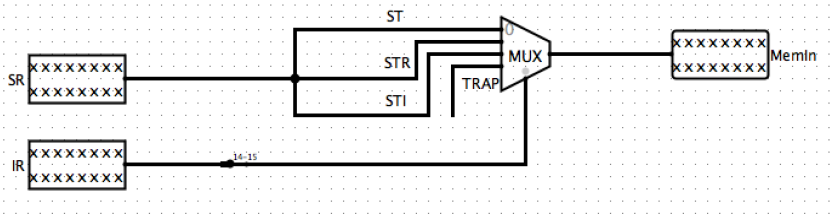

```
.ORIG x3000

        AND R0,R0,0
        BRz fun1
        ADD R0,R0,10
        NOP

fun1:   ADD R0,R0,1
        BRp fun2
        NOP

fun2:   ADD R0,R0,1
        BRz fin
        ADD R0,R0,-3
        BRn fun3
        NOP

fun3:   ADD R0,R0,-1
fin:    NOP

.END
```

À chaque top d'horloge, le compteur PC est incrémenté. C'est là où sont effectués les sauts d'adresses
Si le codeOP de l'instruction est BR, alors le compteur PC est additionné par un offset de 9 bits [8,0] codé dans l'instruction BR ou additionné par la BaseR, c'est à dire le contenu de l'un des registres codé sur 3 bits [8,6].
L'instruction BR peut être conditionnelle ou inconditionnelle, s'il est conditionnel il s'attend à recevoir le signal TestNZP, sans cela, le branchement ne sera pas pris en compte.
Si le codeOP de l'instruction est JMP, il fait un saut à l'adresse contenu dans la BaseR codé dans l'instruction sur 3 bits [8,6].
Si le codeOP de l'instruction est JSR ou JSRR qui ont le même codeOP, on va d'abord mettre le compteur PC incrémenté dans le registre R7, on fera alors la distinction entre les deux sauts de sous-routine par le bit [11] de l'instruction :

- Si le bit [11] vaut 0, alors l'instruction est JSRR et le compteur PC vaudra le contenu de BaseR codé dans l'instruction sur 3 bits [8,6].

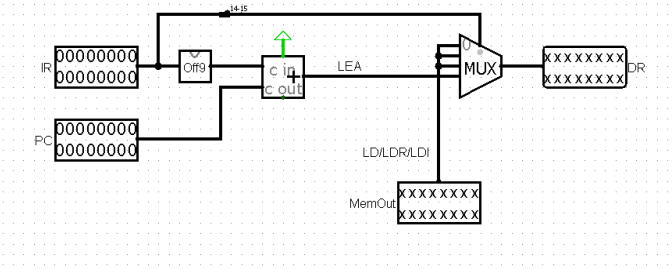
- Sinon le bit [11] vaut 1, et donc l'instruction est JSR, qui fait la somme du compteur PC incrémenté et du Offset codé dans l'instruction sur 11 bits [10,0]

MemIn



Si le codeOP de l'instruction est un rangement, il va être alors stocké en mémoire de la RAM.

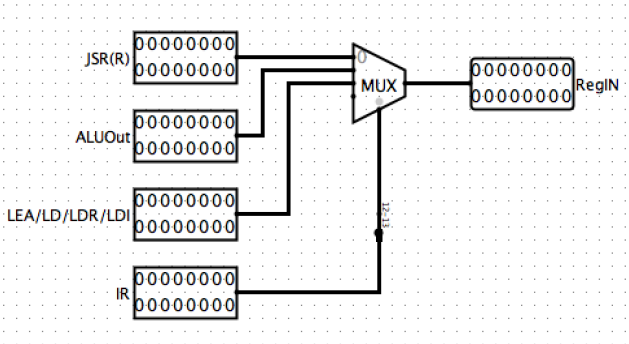
LOAD




```
.ORIG x0000
       LEA R0,cons2     %R0<-3
       NOP

cons1:  .FILL 14
cons2:  .FILL 15
cons3:   .FILL 16

.END
```

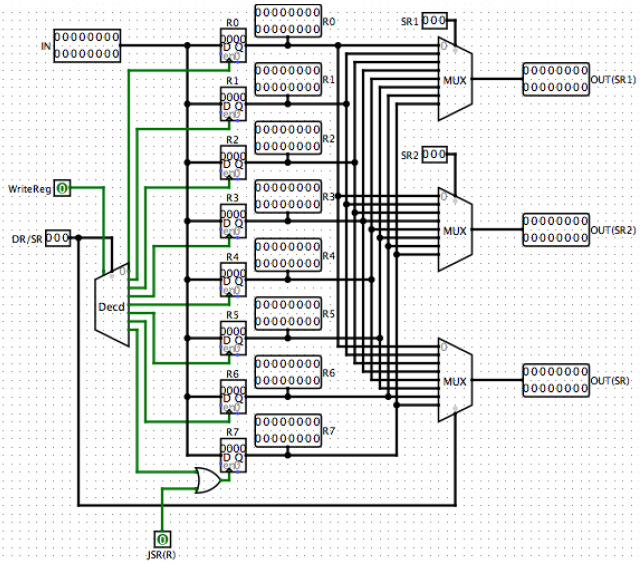
Si le codeOP de l'instruction est un chargement, le compteur PC est additionné avec un Offset de 9 bits [8,0] si l'instruction est LEA.
Sinon pour un LD/LDI/LDR, cela renvoie la sortie de la mémoire de la RAM (MemOUT).

RegCtrlIn



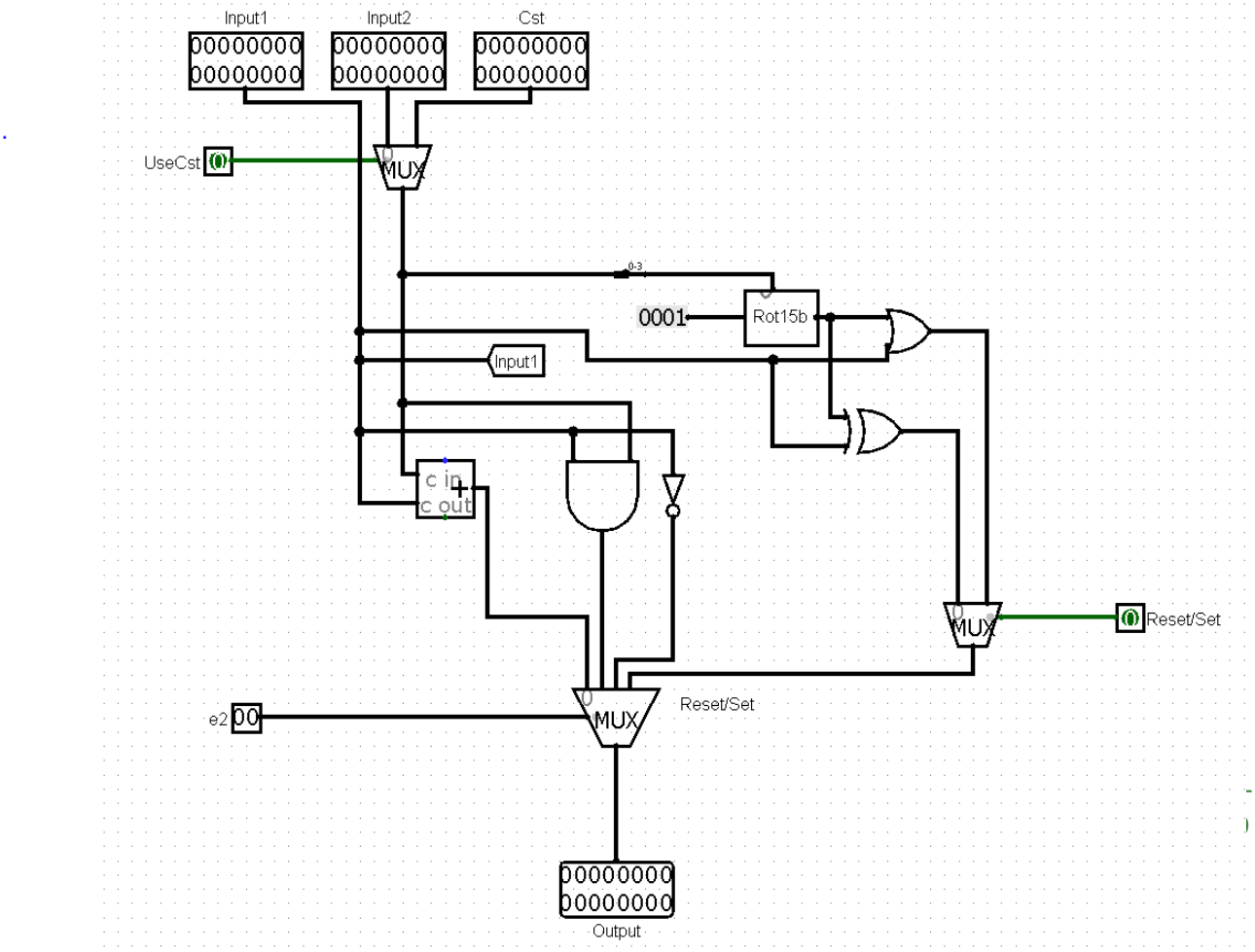
À partir du codeOP de l'instruction elle va choisir avec le multiplexeur quel signal sort, c'est à dire si c'est l'entrée de type arithmétique, chargement ou saut d'instruction.

Registres




```
.ORIG x3000

        AND R0,R0,0
        LEA R1,fun
        JSRR R1
        ADD R0,R0,1
        BRnzp fin
        NOP

fun:    ADD R0,R0,1
        RET
fin:    NOP

.END
```

Quant au banc de registres, il a été légèrement modifié lors de l'ajout de JSR(R), en effet le DecodeIR renvoie un signal « JSR(R) » si le codeOP de l'instruction est JSR(R) le signal va alors être contrôlé en fonction du signal de l'horloge. Il faut qu'elle soit sur l'exécution de l'instruction (EXEC == 1 & JSR(R) == 1). Une fois ces deux conditions validées, elle pourra permettre l'écriture dans le registre R7.



ALU

On peut faire un XOR et un NOT avec le même codeOP.Pour le set/reset nous avons fait un circuit qui fait une rotation sur 15 bits, qui va determiner la position du bit à changer contenu dans les 4 bits de poids faible SR2. Puis en fonction de l'instruction set/reset on va faire respectivement OR ou XOR avec le registre SR1 pour modifier le bit.

Conclusion

Nous voici à la fin de ce rapport, vous trouverez ci-dessous une annexe contenant un glossaire et une petite bibliographie.
Ce projet nous aura permis d'avoir une première approche de la construction d'un processeur.

Annexe

Glossaire

- LC-3 : Mini-processeur

Bibliographie

http://www.wikipedia.org
http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/Cours/LC3/