Rapport Circuits et architecture des ordinateurs

Auteurs : Jérôme Tonnellier et Amina Maimaiti

I – Implémentation

1 - Module DecodeIR

L'entrée de ce module est simplement IR.

On décode simplement la famille d'instruction grâce au bits 12 et 13 de l'entrée IR (famille Jump, Arith, Load ou Store).

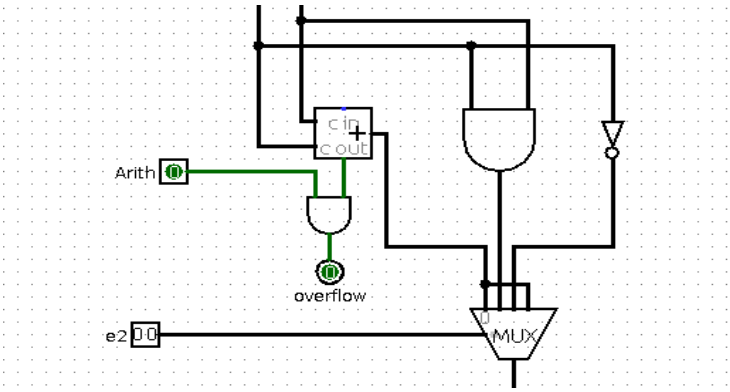
Une fois que nous avons détecté une instruction de la famille Load ou Arith nous pouvons activer la sortie WriteReg afin d'autoriser l'écriture dans le registre adéquat (registre DR du module Registres).

Nous détectons JSR ou JSRR grâce au Jump et au décodage des bits 15 et 14 (respectivement 0 et 1).

Nous détectons également LDI ou STI grâce à, d'une part, la sortie Store et Load et d'autre part, au décodage des bits 15 et 14 (respectivement 1 et 0).

Chaque point décrit ci-dessus représente une sortie dans ce module.

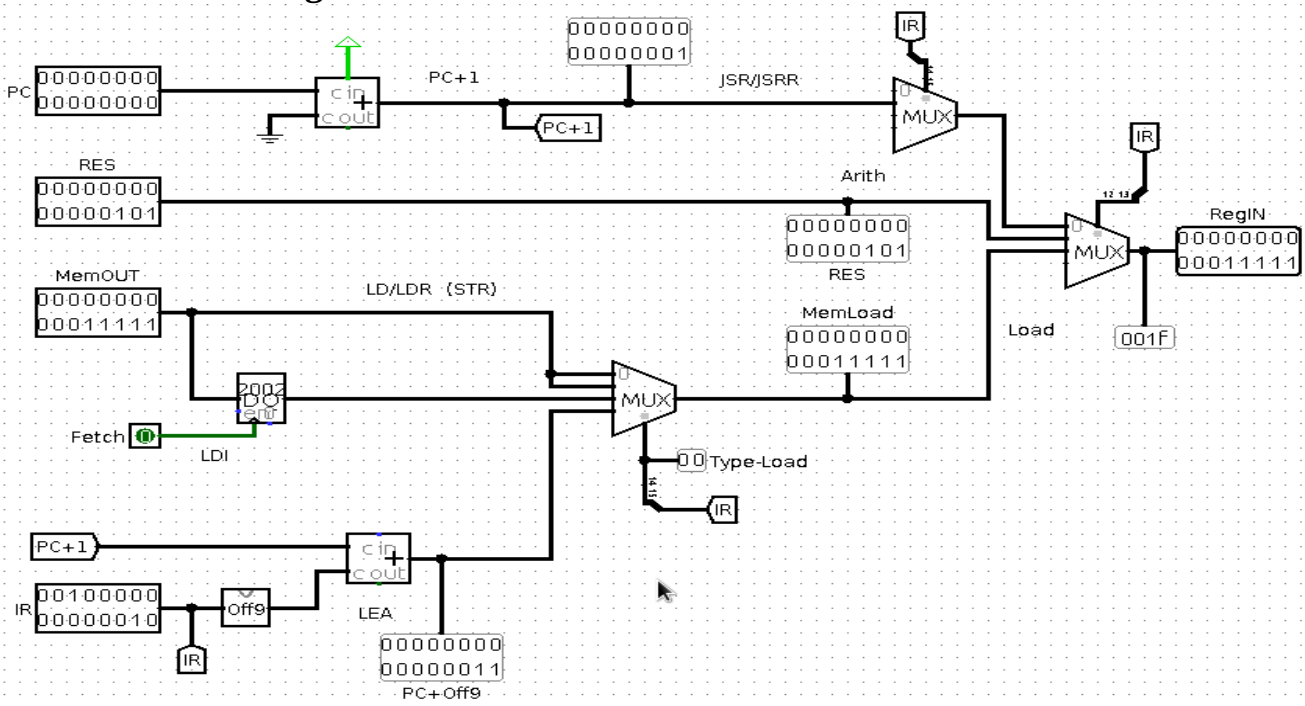
2 – Module ALU



Pour ce module, nous avons seulement ajouté l'opération ADC. Le calcul est le même pour la commande ADD, donc nous avons fait une dérivation sur la sortie de celle-ci.

En plus nous avons détecté un overflow à partir de la retenue signalée par l'additionneur et par le fait qu'on exécute bien une opération Arith.

3 - Module créé RegIN



Ce module nouvellement créé permet d'appliquer les instructions de type Load, Arith ou encore JSR ou JSRR aux registres adéquats. En clair, nous devons calculer toutes les opérations ci-dessus.

Pour une opération provenant de l'ALU, on récupère seulement la valeur RES en entrée dans ce module.

Pour une opération de la famille Load, les cas diffèrent :

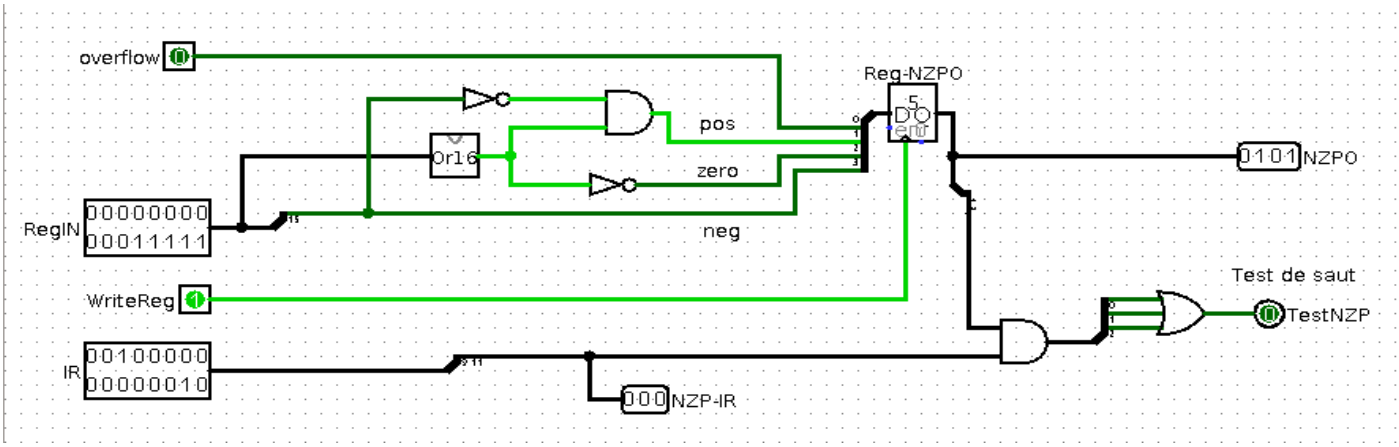
- Pour un LEA, le résultat est l'ajout du PC avec le PCOffset9.
- Pour un LD et LDR, le résultat est récupéré à partir de la sortie mémoire(MemOUT).
- Pour un LDI, on récupère également la sortie mémoire. La seule différence est qu'on stocke dans un registre cette valeur quand le Fetch est activé.

Ces opérations sont sélectionnées par un multiplexeur selon le type de Load.

Pour une opération JSR ou JSRR, on récupère seulement le PC ajouté de 1.

Puis une fois que ces trois types d'opérations sont effectuées, on les sélectionne à l'aide d'un multiplexeur (par rapport aux bits 12 et 13 de IR). On se retrouve avec la bonne valeur (RegIN) à insérer dans le registre DR concerné (module Registres).

4 - Module NZP



Dans ce module, nous testons la valeur de l'entrée RegIN (sortie provenant du module RegIN).

On regarde si cette valeur est positive, négative ou égale à zéro.

Pour tester si cette valeur est négative on regarde le bit 15 de RegIN.

Pour tester le zéro, on utilise le module Or16 (déjà implémenté). Si la sortie de celui-ci donne le bit 0, il n'y a aucun bit de valeur 1 dans RegIN, alors on peut en déduire que RegIN vaut 0.

Pour le test positif on récupère la valeur contraire du test de négation et la négation du test pour zéro (car le nombre doit être le contraire d'un nombre positif et en même temps non égal à zéro).

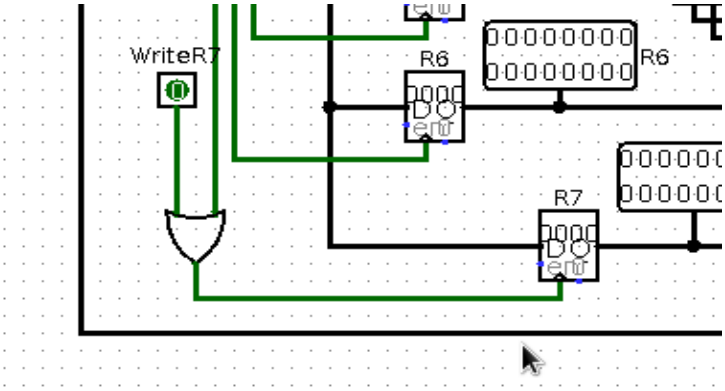
Nous pouvons donc créer la sortie nzpo, (pour le bit de 1 à 3) le nzp correct.

Nous avons ajouté l'entrée overflow provenant du module ALU que nous avons intégré dans la sortie nzpo au bit 0.

La valeur nzpo est stocké dans un registre et mis à jour à chaque fois que le WriteReg est activé.

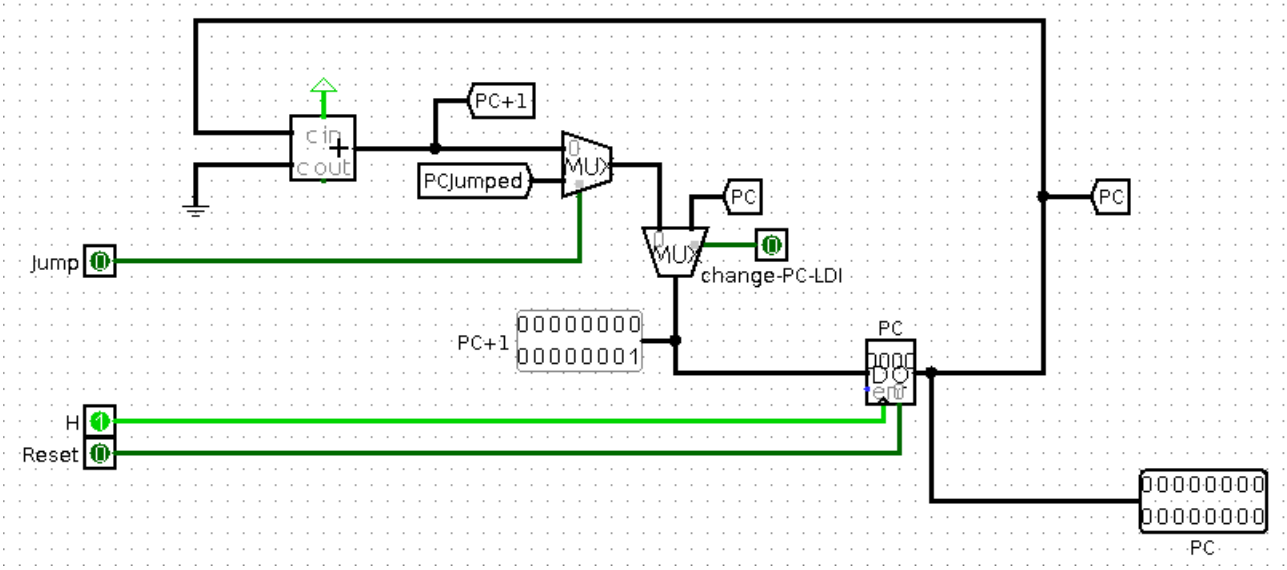
Pour tester si la valeur du nzp calculée correspond au nzp de l'IR, on rassemble ces deux valeurs par un opérateur and. On regarde ensuite si l'un des trois bits est vrai. Si c'est le cas, on active l'autorisation du saut pour BR grâce a la sortie TestNZP (qui s'appelle condition dans le circuit principal).

5 - Module Registres



Pour ce module, la seule modification apportée est au niveau du registre R7. On va faire en sorte d'activer l'écriture de ce registre lors d'un JSR ou JSRR grâce à l'entrée WriteR7. WriteR7 est activé quand le module DecodeIR détecte un JSR ou JSRR et au moment où l'Exec est activé. WriteR7 est visible à la sortie de DecodeIR.

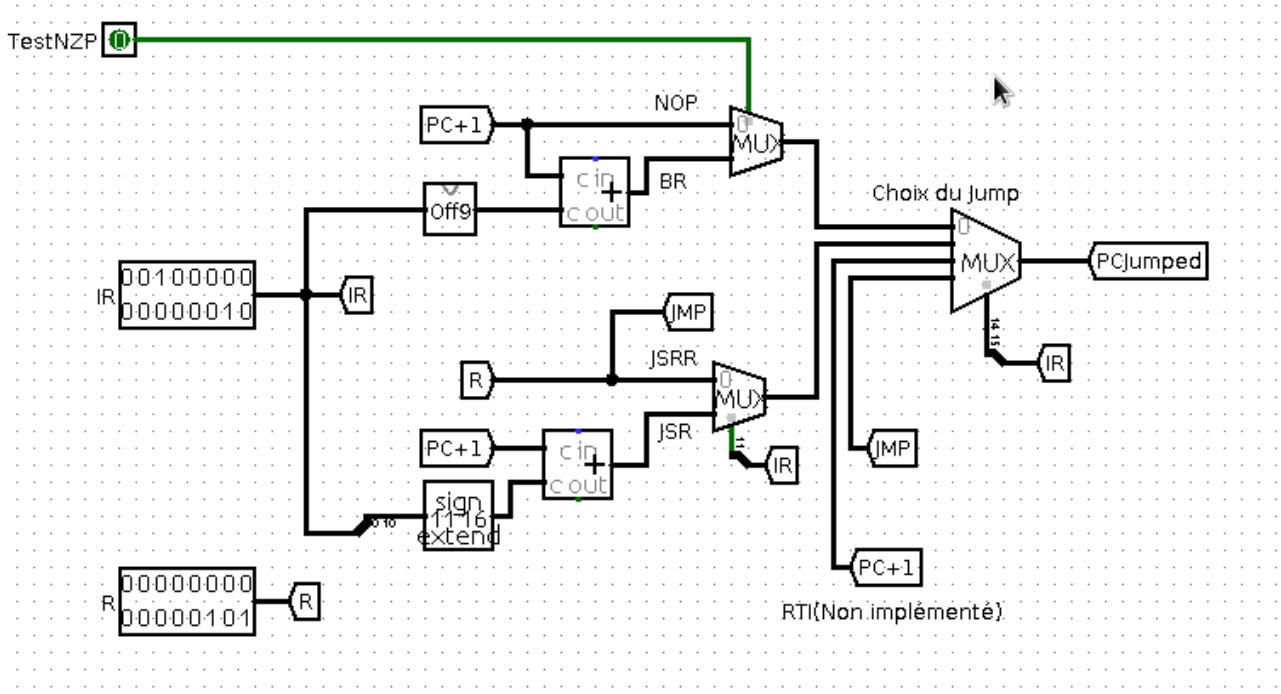
6 - Module RegPC



Dans le module initial, à chaque fois que Exec s'active, on passe de PC à PC + 1.

Nous avons une entrée change-PC-LDI qui s'active lorsque l'on se retrouve avec une instruction LDI ou STI. Une fois activée, elle permet de bloquer le PC.

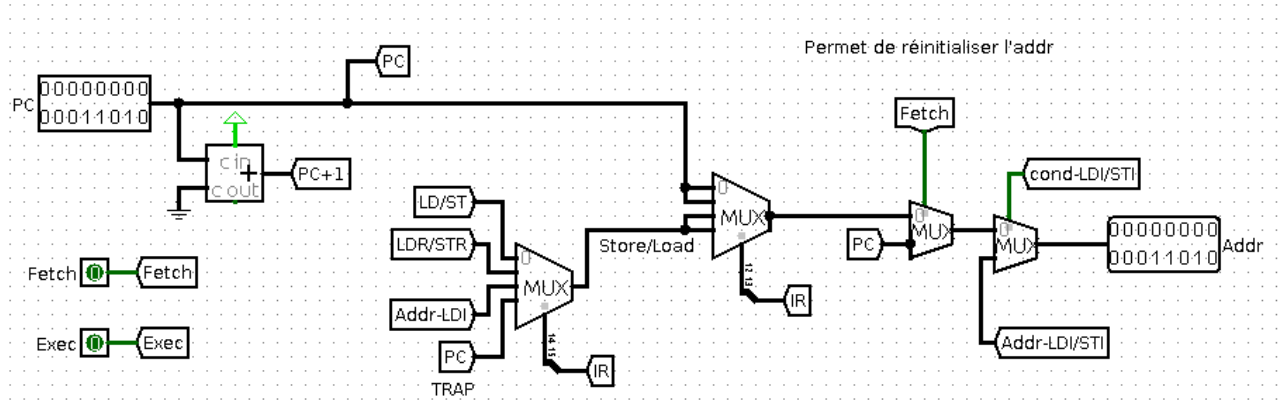
Finalement, nous avons un premier multiplexeur qui permet de faire passer une valeur calculée par une instruction Jump. Bien sûr, on peut récupérer ce résultat seulement quand l'entrée Jump est activée. Ensuite, avant de renvoyer le résultat Jump, on choisi le type du Jump (selon les bits 15 et 14) grâce au multiplexeur.



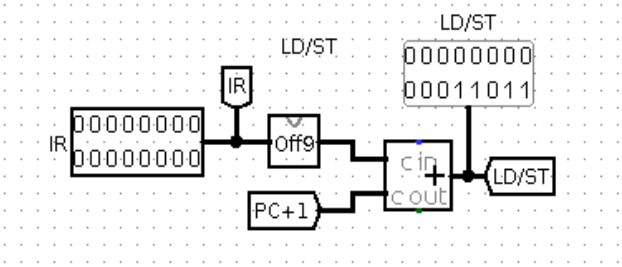
Il y a donc 4 opérations :

- Pour le premier cas, si le TestNZP est activé, le multiplexeur laisse passer la valeur de BR, sinon on ne fait rien (NOP). Le résultat du BR est l'addition du PC avec PCoffset9.
- Pour le deuxième cas, on renvoie soit, le contenu de BaseR (entrée R) pour la commande JSRR, soit, l'addition de PC et PCoffset11. Un multiplexeur sélectionne l'un ou l'autre.
- Pour le troisième cas, on ne fait rien (commande RTI non implémentée).
- Pour le quatrième cas, on renvoie la valeur de BaseR.

7 - Module GetAddr



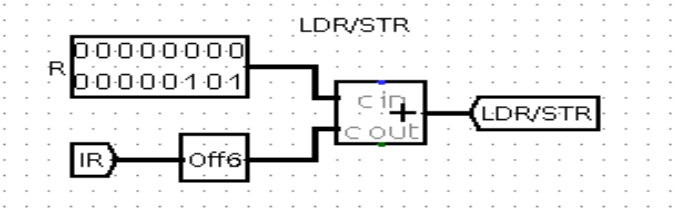
Pour les instructions de type Arith et Jump, l'adresse mémoire Addr n'est pas modifiée.
Sinon, pour les instructions de type Store ou Load, on sélectionne grâce à un multiplexeur les commandes suivantes :

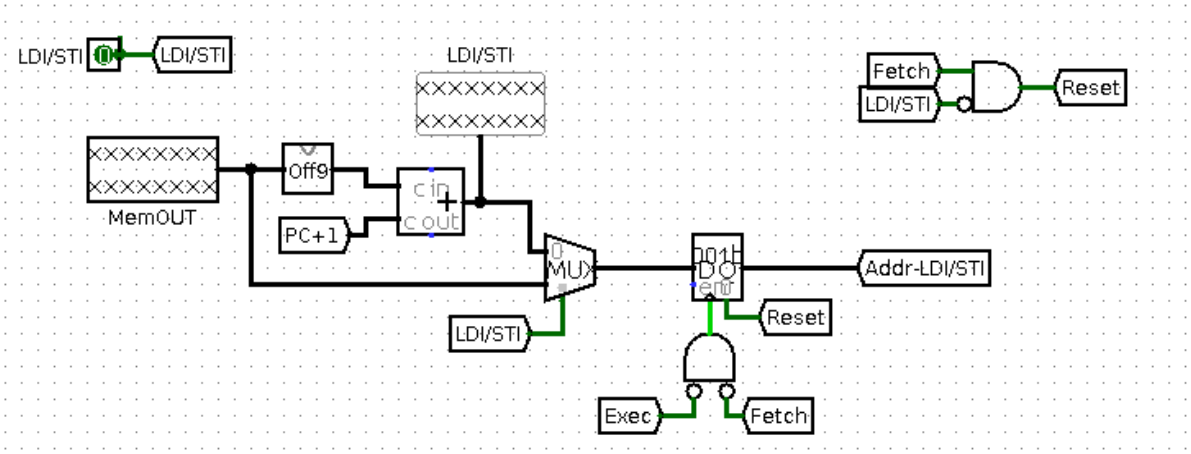


- On commence par LD et ST. On renvoie simplement l'addition du PC avec PCoffset9.

- Ensuite, on a LDR et STR. On renvoie le contenu de BaseR additionné du PCoffset6.

- Pour la commande LEA on ne change rien.





– Les commandes LDI et STI sont gérées sur 8 temps (au lieu de 4 temps pour une instruction classique). Le but de jeu est de récupérer MEM[MEM[PC+PCoffset9]] dans les 6 premiers temps. Pour cela nous nous aidons de MemOUT et d'un registre. MemOUT est l'instruction de sortie de RAM. Sa valeur est récupéré au 1er, 3e et 5e temps. La valeur de MemOUT est stocké dans le registre au 2e, 4e et 6e temps. A chaque fois qu'on récupère la valeur de MemOUT on la stocke ensuite dans le registre. Cette opération est donc effectuée 3 fois à la suite afin de récupérer MEM[MEM[PC+PCoffset9]].

Donc en résumé :
– 1er et 2e temps : on récupère PC+PCoffset9 puis on le stocke dans le registre.
– 3e et 4e temps : on récupère MEM[PC+PCoffset9] puis on le stocke dans le registre.
– 5e et 6e temps : on récupère MEM[MEM[PC+PCoffset9]] puis on le stocke dans le registre.
– Pour les temps 7 et 8, la sortie fin-condition-LDI/STI est activée. Elle permet de réinitialiser L'IR afin de préparer l'instruction suivante.

II – Tests

1 – Le fichier fourni nbocc.asm (source : site du professeur de cours) fourni permet de tester les commandes LEA, LD, LDR et BR.
4 – Le fichier fourni exam-session1-13-14-sujetA.asm teste LD, JSR, BR, STR,LDR et RET.
3 – Le fichier fourni JMP-ST-ADC.asm teste JMP, ST et ADC.
2 – Le fichier fourni JSR-JSRR.asm permet de tester donc JSR et JSRR.
5 – Le fichier fourni LDI-STI.asm teste LDI et STI.