Année 2021-2022

RAPPORT DE PROJET – ENSIBS Cybersécurité du Logiciel – 3e année

PROJET MICROPROCESSEUR « Processeur 8-bits »

Projet réalisé par

Lucas CHAPRON

Projet encadré par

Vianney LAPÔTRE

Table des matières

I. INTRODUCTION

Dans le cadre de ma première année du cycle ingénieur en Cybersécurité du Logiciel à l'ENSIBS, il nous est proposé un projet de 11h nous permettant de mettre en pratique nos connaissances et nos compétences pour le développement d'un premier processeur pour explorer le monde de l'hardware.

II. DÉVELOPPEMENT

1. CONTEXTE

Le but de ce projet est de construire un *petit processeur 8 bits* à base d'éléments logiques et séquentiels dans le but d'appréhender les mécanismes de base du fonctionnement interne d'un processeur de type RISC. Ce petit processeur simplifié ne sera pas pipeliné (une instruction sera entièrement traitée en un cycle). De plus, l'accès à la mémoire sera réalisé en un seul cycle, simplifiant le contrôle du processeur (cas de la mémoire cache parfaire : 0 miss).
Pour ce faire, je vais utiliser le logiciel *Logisim-evolution*.

2. CONCEPTION DES UNITÉS LOGIQUES

Pour cette étape, j'ai construit 3 circuits logiques permettant de réaliser les opérations AND, OR et XOR sur des données de 8 bits.
Cependant, on avait plusieurs contraintes :

- On ne pouvait utiliser que des portes logiques à 2 entrées de 1 bit
- On devait intégrer ces 3 circuits dans un module nommé « UL » (unité logique). Ce module possède 2 entrées de 8 bits chacune et 3 sorties de 8 bits également.

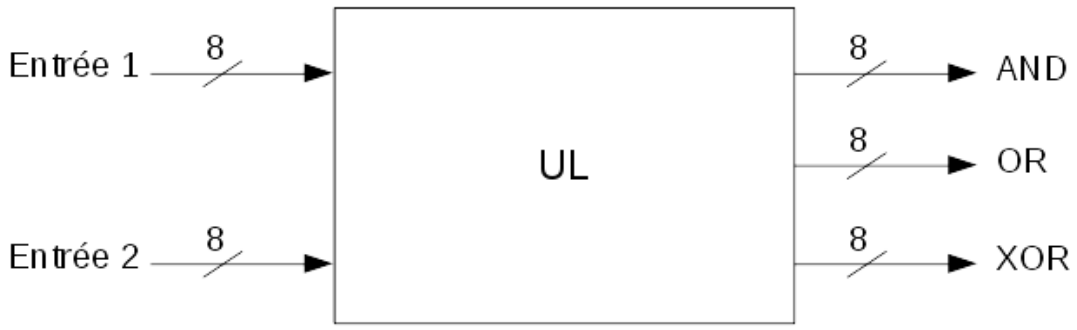

Figure 1 : SCHÉMA UL (Unité Logique)

Pour réaliser mes circuits logiques je n'ai pas réalisé de table de vérité car très long à la main (256 états) je me suis basé sur la logique de fonctionnement. Pour faire une opération entre 2 octets il suffit de faire l'opération sur leurs bits qui sont à la même position. Je m'explique si je veux faire un XOR entre deux octets il faut que je fasse un XOR entre les deux bits qui ont la position 0 puis ceux qui ont la position 1 et ainsi de suite jusqu'à la position 7. Ce qui me donne les portes logiques suivantes :

Pour le AND :

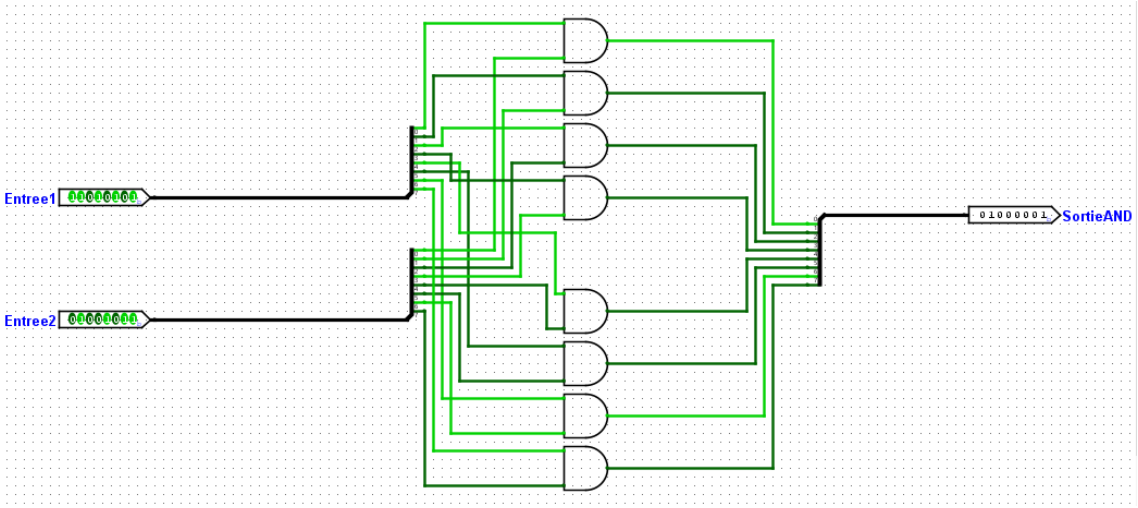

Figure 2 : AND 8-bits

Pour le OR :

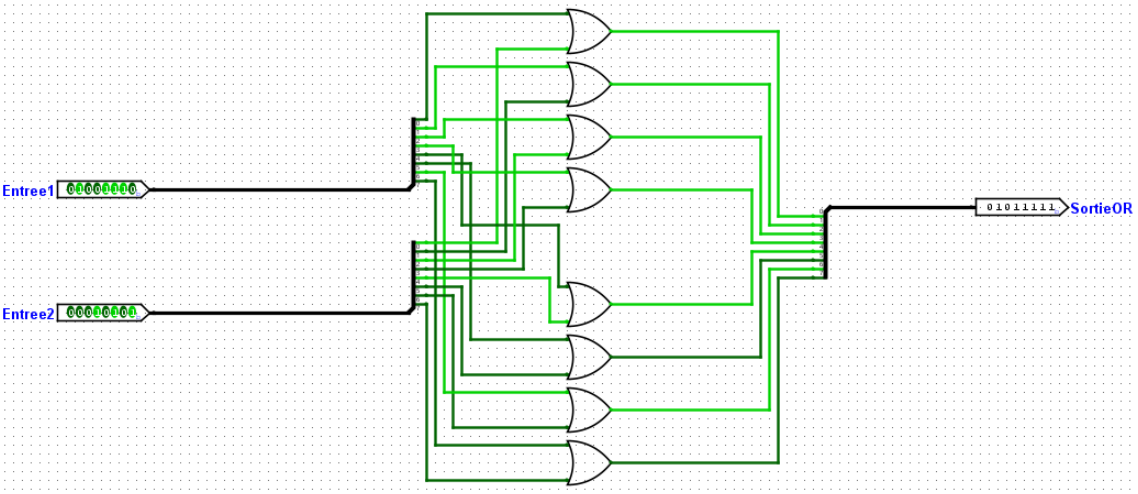

Figure 3 : OR 8-bits

Pour le XOR :

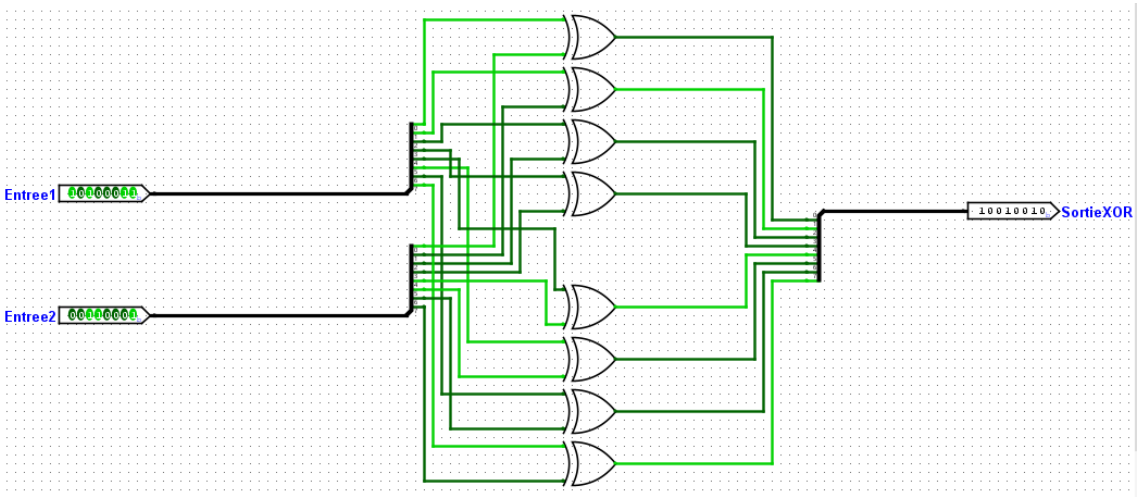

Figure 4 : XOR 8-bits

On obtient alors l'UL voulu :

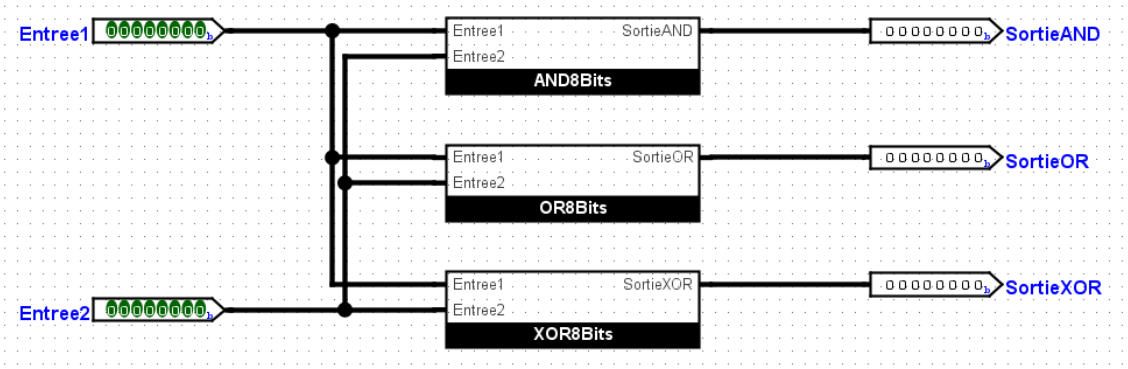

Figure 5 : UL (Unité Logique)

3. CONCEPTION D'UN ADDITIONNEUR

Pour cette étape, je dois construire un additionneur 8-bits. Pour ce faire, je commence par faire un additionneur 1-bit complet. Ce module me servira de base pour construire un additionneur n-bits (8-bits dans notre cas). Ce module se présente comme suit :

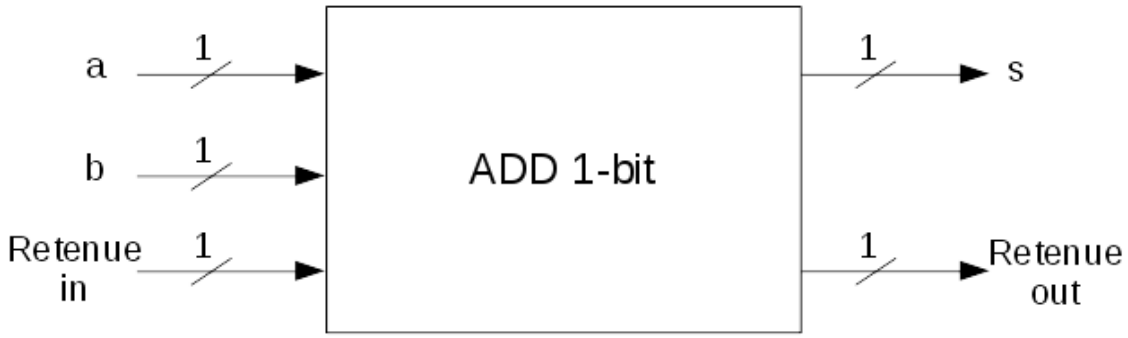

Figure 6 : SCHÉMA Additionneur 1-bit

Pour construire ce circuit, j'ai construit sa table de vérité afin d'en extraire les deux équations définissant ses sorties.

Rin	a	b	Somme	Rout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 7 : Table de vérité additionneur 1-bit

On obtient alors les équations suivantes que je simplifie par simple factorisation :

- Somme = f (R_{in}, a, b) = ($\overline{R_{in}}$ * $\bar{a}$ * b) + ($\overline{R_{in}}$ * a * $\bar{b}$) + (R_{in} * $\bar{a}$ * $\bar{b}$) + (R_{in} * a * b)
  = $\overline{R_{in}}$ * (($\bar{a}$ * b) + (a * $\bar{b}$)) + R_{in} * (($\bar{a}$ * $\bar{b}$) + (a * b))

- R_{out} = f (R_{in}, a, b) = ($\overline{R_{in}}$ * a * b) + (R_{in} * $\bar{a}$ * b) + (R_{in} * a * $\bar{b}$) + (R_{in} * a * b)
  = R_{in} * (a + ($\bar{a}$ * b)) + $\overline{R_{in}}$ * (a * b)

Ce qui me permet de réaliser 2 sous-circuits : un pour réaliser la somme et l'autre pour la retenue.
Pour la somme :

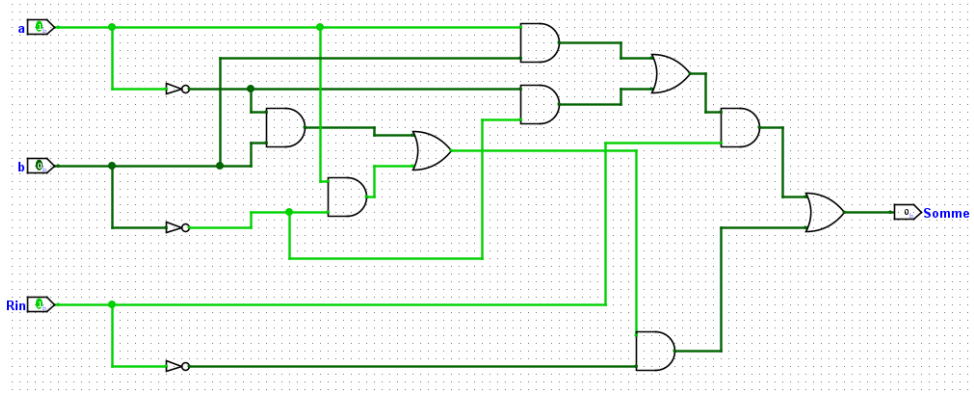

Figure 8 : Somme Additionneur 1-bit

Pour la retenue de sortie :

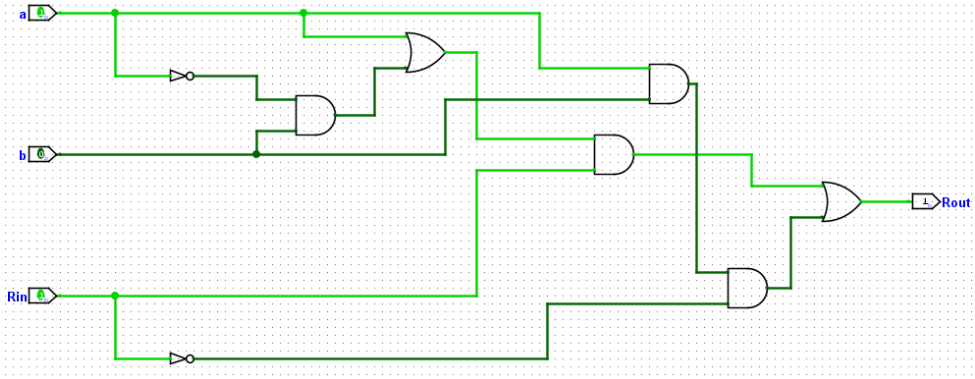

Figure 9 : R_{out} Additionneur 1-bit

À partir de ces deux circuits je peux maintenant réaliser l'additionneur 1-bit qui me servira après.

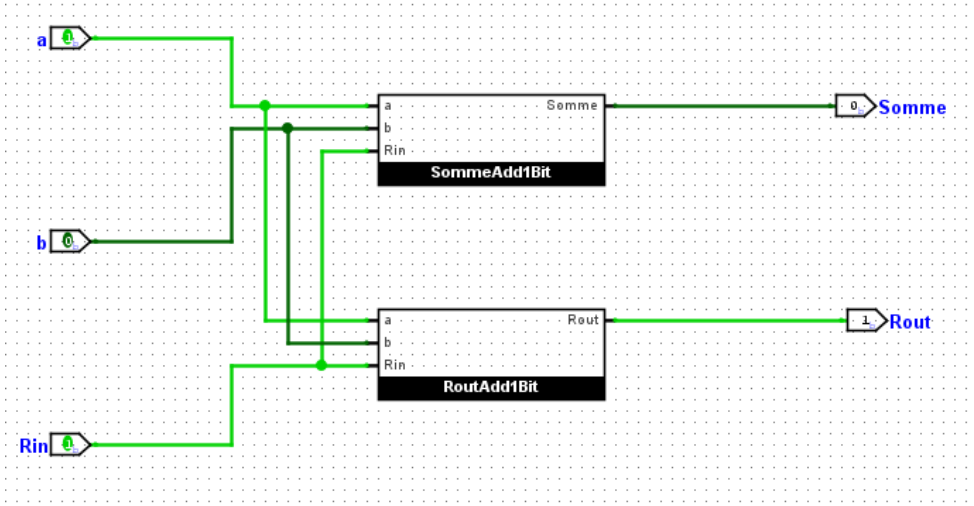

Figure 10 : Additionneur 1-bit

Dès lors je peux réaliser un additionneur 8-bits tel que :

Figure 11 : SCHÉMA Additionneur 8-bits

Pour réaliser ce circuit, je mets en cascade 8 additionneurs 1 bit en récupérant les retenues :

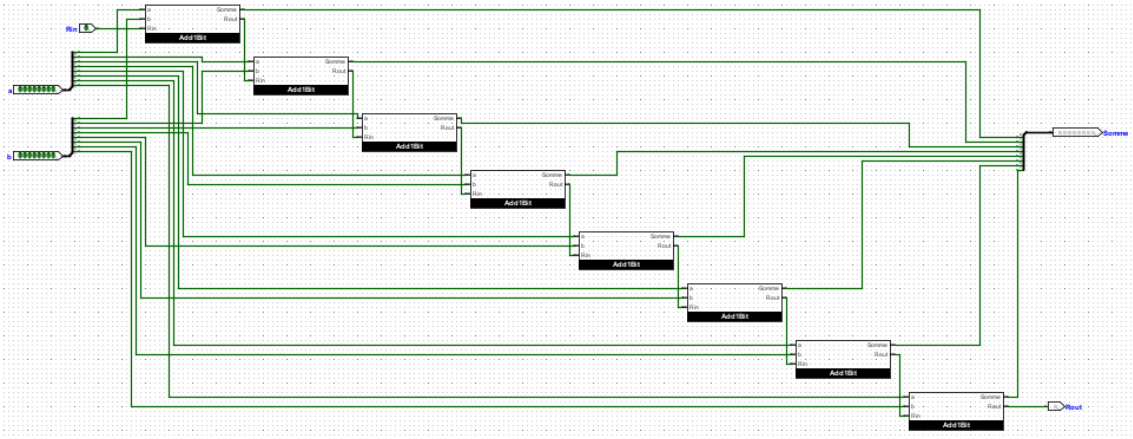

Figure 12 : Additionneur 8-bits

4. CONCEPTION DE L'UAL

L'UAL du processeur sera le cœur opératif du processeur. Dans mon cas, cette unité est capable de réaliser 4 opérations. 3 opérations logiques et 1 opération arithmétique (addition). Toutes ces opérations sont réalisées sur des données de 8 bits. L'opération à réaliser est définie via un signal dédié comme indiqué sur la figure ci-dessous :

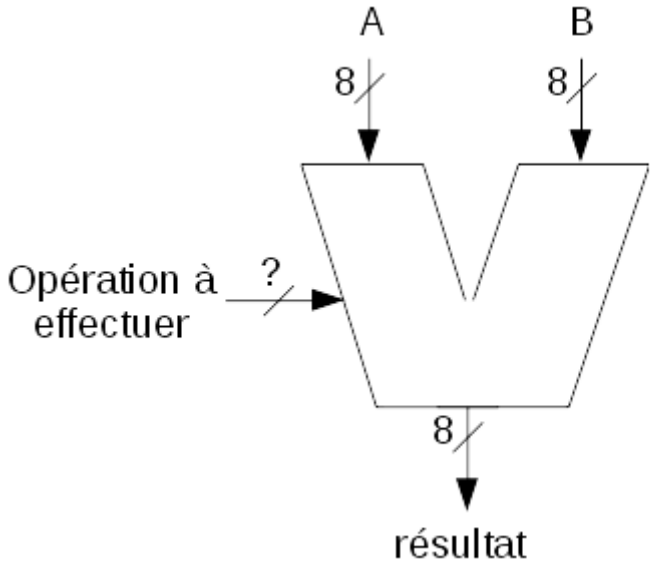

Figure 13 : SCHÉMA UAL (Unité Arithmétique et Logique)

Pour le réaliser il suffit de multiplexer les sorties pour avoir le résultat voulu :

- 00 : Addition 8-bits
- 01 : OR 8-bits
- 10 : AND 8-bits
- 11 : XOR 8-bits

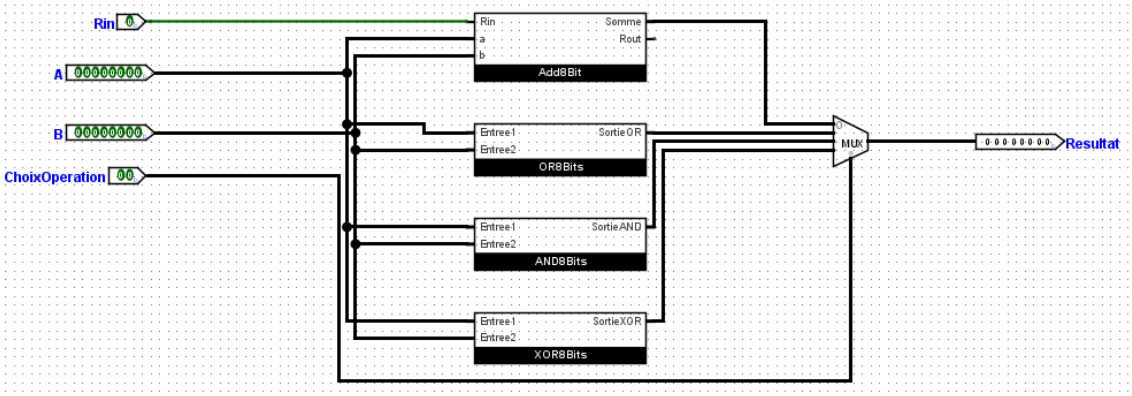

Figure 14 : UAL (Unité Arithmétique et Logique)

5. CONCEPTION D'UN BANC DE REGISTRES

Pour cette étape, je vais créer 4 registres permettant de stocker des valeurs de 8 bits. Dans un premier temps, je vais créer un module registre 8 bits que je vais ensuite dupliquer pour créer le banc de registres complet.

Le registre est réalisé comme suit :

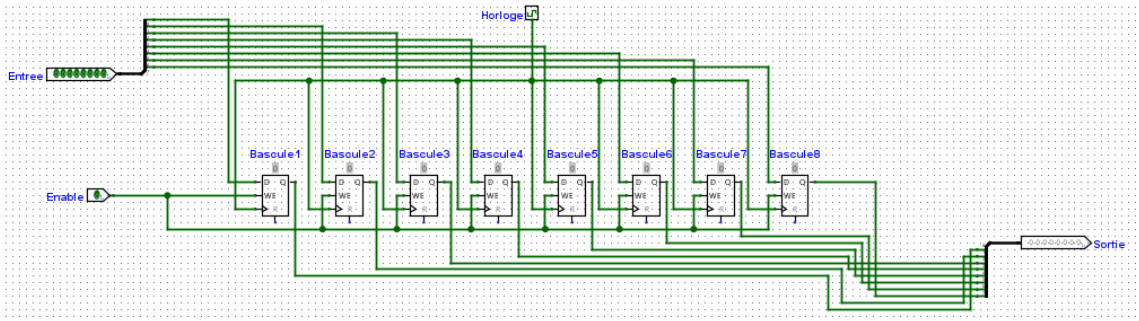

Figure 15 : Registre 8-bits

Pour le banc de registre, il a une entrée pour deux sorties (A et B) et le choix d'écrire ou non (enable). Donc j'ai dû utiliser des multiplexeurs pour avoir ce choix-là. Il me faut donc 2 bits pour le choix du registre de sortie que je dois multiplier par 2 car on a 2 sorties. Mais également 2 bits pour le choix de la destination du enable.

- Les deux premiers bits sont pour le choix du registre d'écriture suivant le choix de registre ci-dessous
- Les bits 2 et 3 sont pour la sortie A suivant le choix de registre ci-dessous
- Les deux bits de poids faible sont pour la sortie B avec le même choix de registre

00 : Choix de R1
01 : Choix de R2
10 : Choix de R3
11 : Choix de R4

Donc si "SelectionRegistre" vaut : 010011 alors j'écris dans R2, la sortie A aura la valeur de R1 et B celle de R4.

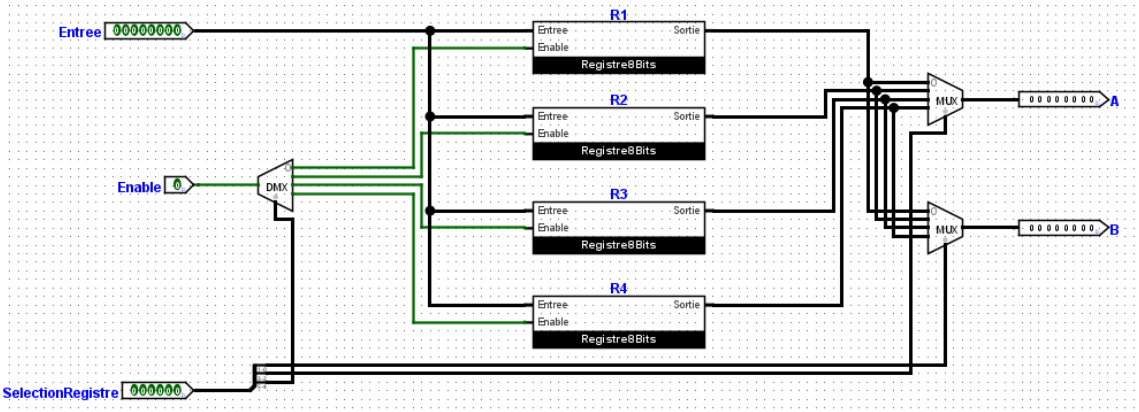

Figure 16 : Banc de Registres

6. CONCEPTION D'UNE UNITÉ DE TRAITEMENT COMPLÈTE

Cette étape a pour but de tester le fonctionnement du couple banc de registre - UAL. Je relie le banc de registre avec l'UAL tel que :

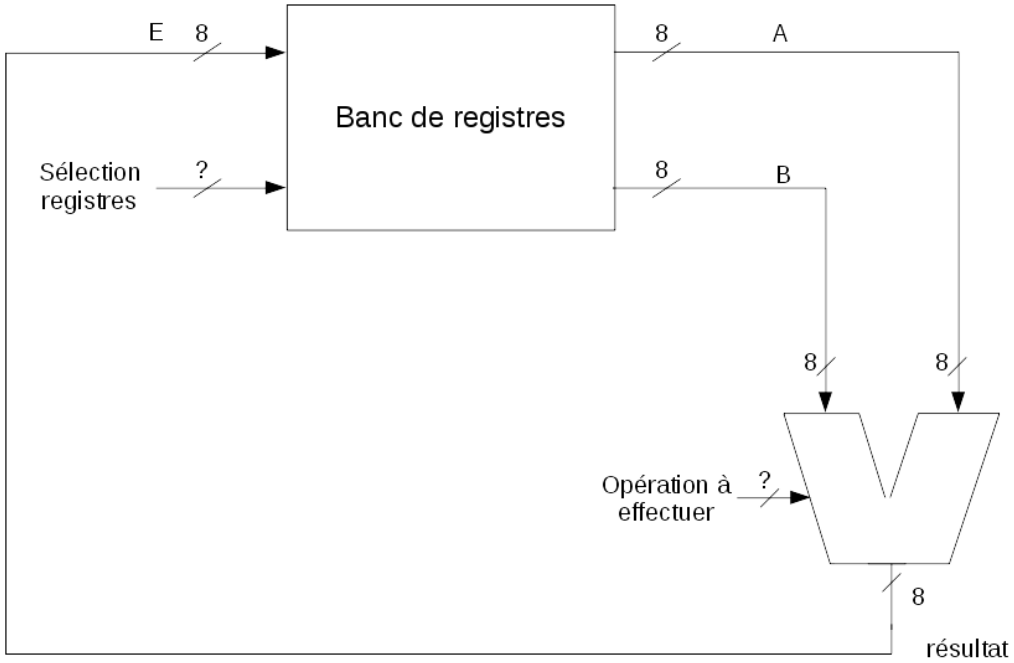

Figure 17 : SCHÉMA Unité de Traitement

Il faut donc pouvoir initialiser le banc de registre si je veux que l'unité de traitement puisse faire quelque chose donc on obtient le circuit suivant :

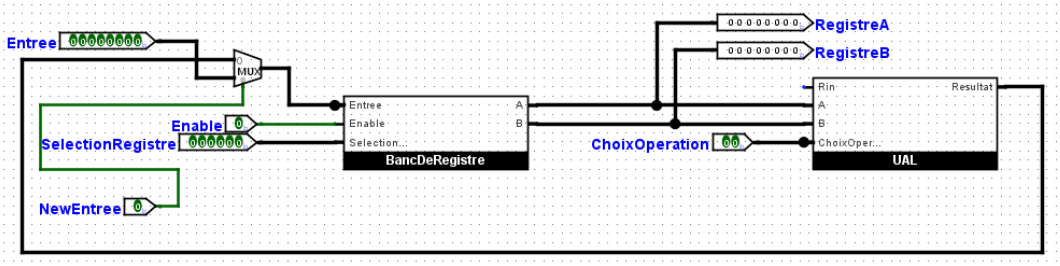

Figure 18 : Unité de Traitement

Le "NewEntree" sert à savoir si la valeur d'entrée est une valeur venue de l'extérieur (1 bit) ou celle sortant de l'UAL. Les sorties "RegistreA" et "RegistreB" ne servent que plus tard lors de l'ajout de mémoire RAM. Ici elles ne serviront pas.

7. INTÉGRATION D'UNE MÉMOIRE PROGRAMME ET DU POINTEUR PROGRAMME

Cette étape a pour but d'implémenter un processeur possédant une mémoire ROM contenant des instructions qui seront décodées dans le but de piloter mon processeur selon le modèle ci-dessous :

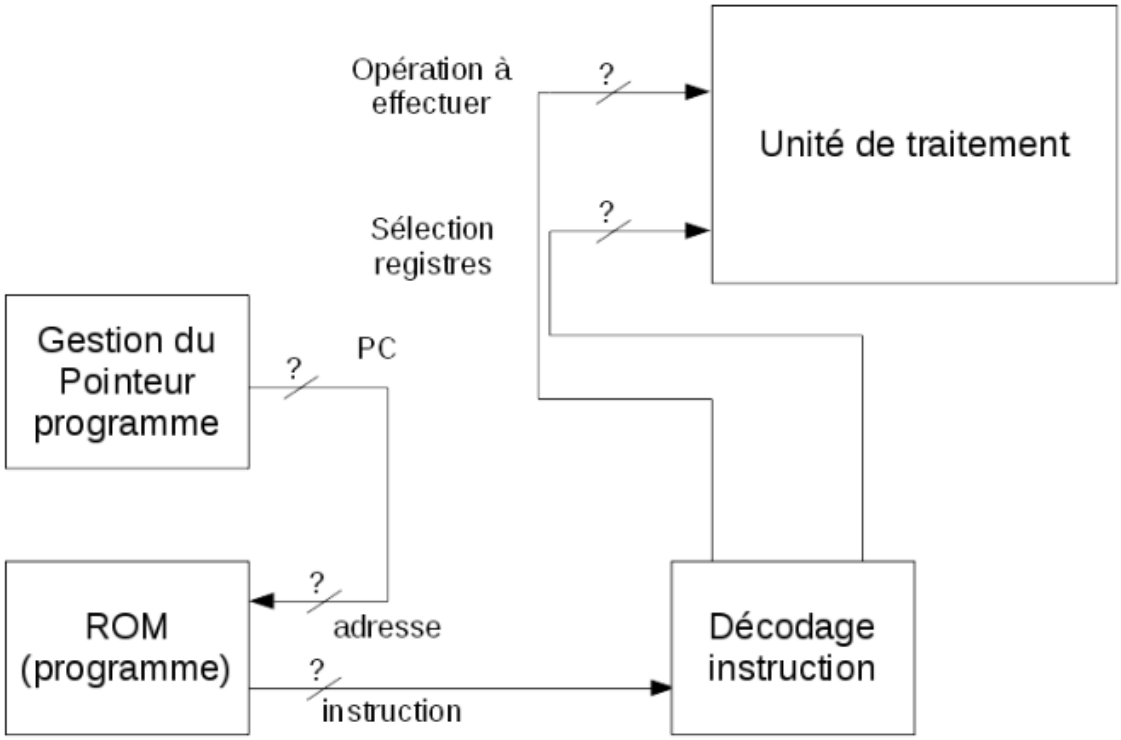

Figure 19 : SCHÉMA Processeur sans RAM

Pour ce faire, il me faut créer un gestionnaire de pointeur programme qui me permet de savoir quelle instruction exécuter en envoyant l'adresse dans la ROM, des règles d'instructions pour que je puisse écrire des instructions dans la ROM et un décodeur d'instruction pour que les instructions sortant de la ROM puissent être correctement interprétées.

Pour la gestion du pointeur programme, puisqu'il n'y a pas d'instructions de branchement, celle-ci est relativement simple. Elle nécessite la génération d'une adresse permettant de lire l'instruction suivante dans la mémoire. A chaque cycle, une instruction doit être lue depuis la mémoire programme. La première instruction du programme se trouvera à l'adresse '0x00' de la mémoire. Pour ce faire, il nous faut un endroit où stocker le pointeur du programme et un additionneur pour pouvoir passer d'une instruction à une autre. D'où le gestionnaire de pointeur programme suivant :

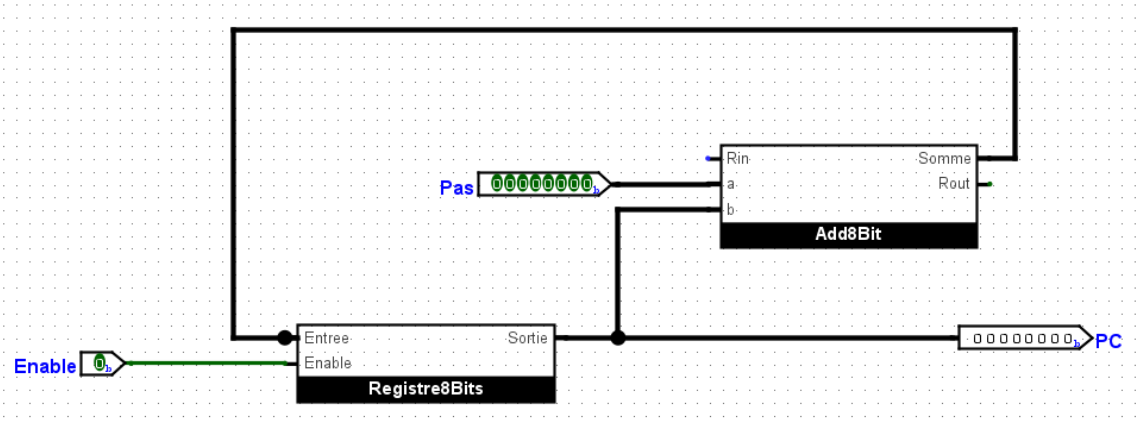

Figure 20 : Gestion de Pointeur Programme

Pour les règles de la ROM :
Il faut gérer l'opération à choisir (UAL : 2 bits), le choix de registre & d'écriture (Unité de traitement : 6 bits), des nombres à manipuler (entrée : 8 bits pour plus de facilité), le enable (1 bit) et l'information comme quoi il y a une nouvelle entrée car sinon on va faire des opérations avec des 0 (1 bit).
J'ai donc besoin de 18 bits.
Je définis une instruction tel que :
0 0 00000000 000000 00

- Le premier bit est pour savoir s'il y a une nouvelle entrée
- Le deuxième bit est pour le enable donc est ce qu'on a besoin d'écrire
- Les 8 prochains bits sont les nombres à manipuler
- Les 6 bits d'après sont pour le choix de registre à écrire et ceux à sortir
- Les 2 derniers bits sont pour l'opération à faire

Je veux faire l'opération 3*4=12 donc je vais sommer 4, 3 fois

1/ Je fais venir 4 que stocke dans le R2 (somme avec un registre à 0 (R4))
2/ Je fais une somme entre le R2 et R1 (à 0 pour l'instant) et je stocke le résultat dans R1
3/ Je fais une somme entre le R2 et R1 et je stocke le résultat dans R1
4/ Je fais une somme entre le R2 et R1 et je stocke le résultat dans R1

En binaire cela correspond à :
1/ 1 1 00000100 010111 00 ->R1=0, R2=4, R3=0, R4=0
2/ 0 1 00000000 000001 00 ->R1=4, R2=4, R3=0, R4=0
3/ 0 1 00000000 000001 00 ->R1=8, R2=4, R3=0, R4=0
4/ 0 1 00000000 000001 00 ->R1=12, R2=4, R3=0, R4=0
Comme la ROM prend des valeurs en hexadécimal je convertis ce qui donne :

1/ 3045C 3/ 10004
2/ 10004 4/ 10004

Pour le décodeur il faut juste séparer les instructions envoyées par la ROM pour bien les répartir sur les bonnes entrées donc on obtient le circuit suivant :

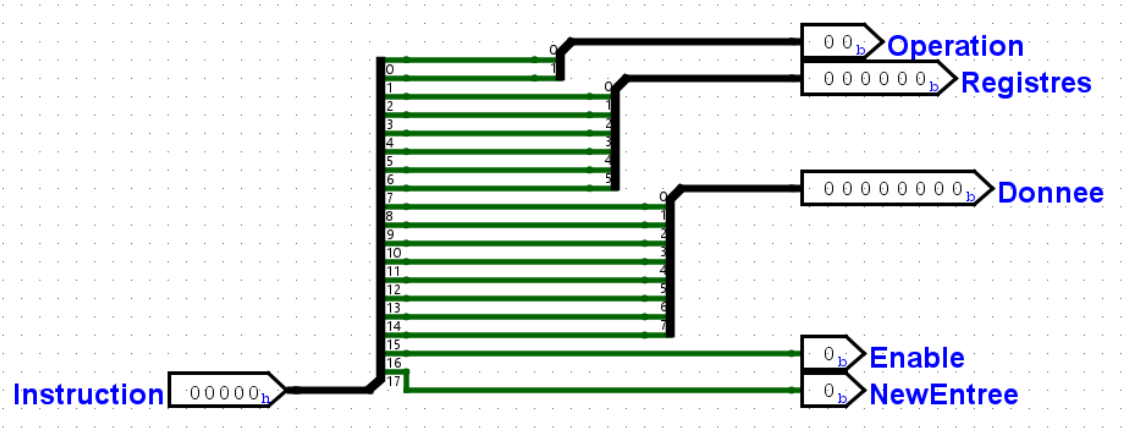

Figure 21 : Décodeur V1

En combinant tous nos circuits j'obtiens un premier processeur, cependant il ne possède, ni mémoire (mis à part le banc de registre de l'unité de traitement) ni branchement possible.

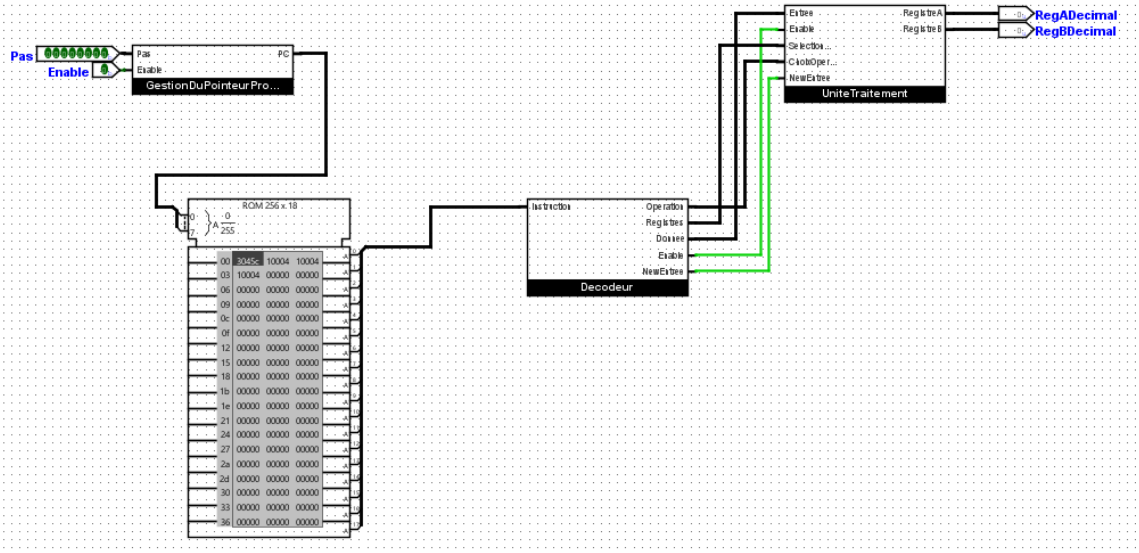

Figure 22 : Processeur étape 7

8. INTÉGRATION D'UNE MÉMOIRE DE DONNÉES

Pour le moment, mon processeur ne contient que très peu d'espace de stockage. En effet, le nombre de registres est limité dans un processeur. Dans le but de palier ce problème, je vais ajouter une mémoire RAM permettant de lire et écrire des données durant l'exécution de mon programme. Le schéma du circuit est tel que :

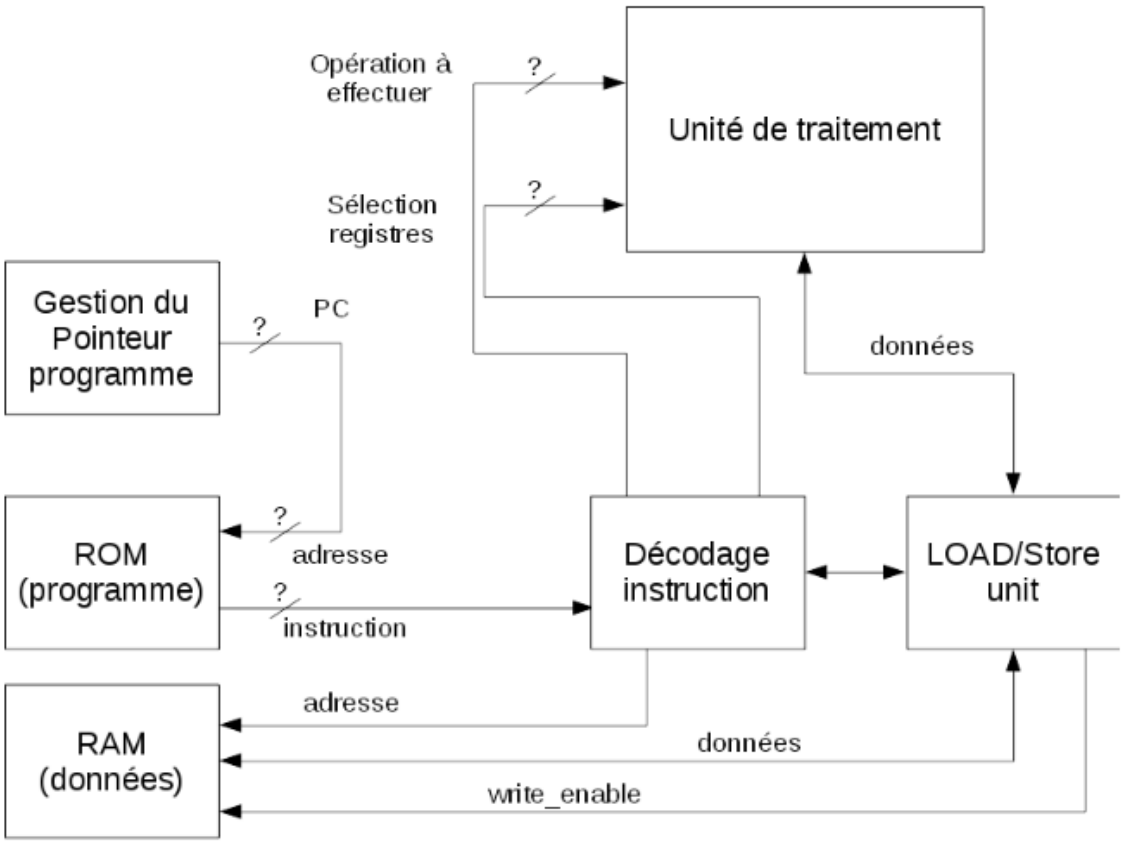

Figure 23 : SCHÉMA Processeur avec RAM

Pour réaliser ceci il me faut modifier les règles de la ROM précédemment établi :
Je dois savoir si je dois load (1 bit) ou store (1 bit). J'ai séparé load et store ainsi si les 2 sont à 0 je fais un load instantanée et si l'un des deux est à 1 alors je lis les 8 bits comme une adresse et non un nombre. J'aurais juste à mettre le enable de l'unité de traitement à 0. Je choisie de faire en sorte qu'il

n'y ait que la valeur qui est stockée dans A qui puisse être store pour plus de simplicité (j'aurais pu mettre un multiplexeur de plus mais ça n'aurait pas apporté beaucoup plus de fonctionnalité à mon processeur). Je me sers des 8 bits pour avoir une adresse cela me fait gagner en taille d'instructions.
J'ai donc un nouveau format d'instruction sur 20 bits :
0 0 0 0 00000000 000000 00

- Le premier bit est pour savoir s'il y a une nouvelle entrée (1=oui, 0=non)
- Le deuxième bit est pour le enable donc est ce qu'on a besoin d'écrire (1=oui, 0=non)
- Le troisième bit est pour savoir si on fait un load (1=oui, 0=non)
- Le quatrième bit est pour savoir si on fait un store (1=oui, 0=non)
- Les 8 prochains bits sont les nombres à manipuler
- Les 6 bits d'après sont pour le choix de registre à écrire et ceux à sortir
- Les 2 derniers bits sont pour l'opération à faire

Je veux faire l'opération 3*4=12 dont le résultat est stocké dans la RAM, je fais 42+42, je load le résultat d'avant pour le XOR avec celui-ci ce qui donne 88.

1/ Je fais venir 4 que stocke dans R2 (somme avec un registre à 0 (R4))
2/ Je fais une somme entre R2 et R1 (à 0 pour l'instant) et je stocke le résultat dans R1
3/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
4/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
5/ Je store R1 dans la RAM à la dernière adresse
6/ Je remets R1 à 0 en envoyant 0 et en sommant avec un registre à 0 (R3)
7/ Je fais venir 42 que stocke dans R2 (somme avec un registre à 0 (R4))
8/ Je fais une somme entre R2 et R1 (à 0 pour l'instant) et je stocke le résultat dans R1
9/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
10/ Je remets R2 à 0 en envoyant 0 et en sommant avec un registre à 0 (R3) (je fais juste attention de toujours avoir un registre à 0 parce que je n'ai pas de reset)
11/ Je load la valeur stockée à la dernière place de la mémoire RAM dans R3
12/ Je fais un XOR entre R3 et R1 (à 84 pour l'instant) et je stocke le résultat dans R1

En binaire cela correspond à :

1/ 1 1 0 0 00000100 010111 00 -> R1=0, R2=4, R3=0, R4=0
2/ 0 1 0 0 00000000 000001 00 -> R1=4, R2=4, R3=0, R4=0
3/ 0 1 0 0 00000000 000001 00 -> R1=8, R2=4, R3=0, R4=0
4/ 0 1 0 0 00000000 000001 00 -> R1=12, R2=4, R3=0, R4=0
5/ 0 0 0 1 11111111 000000 00 -> R1=12, R2=4, R3=0, R4=0
6/ 1 1 0 0 00000000 000010 00 -> R1=0, R2=4, R3=0, R4=0
7/ 1 1 0 0 00101010 010111 00 -> R1=0, R2=42, R3=0, R4=0
8/ 0 1 0 0 00000000 000001 00 -> R1=42, R2=42, R3=0, R4=0
9/ 0 1 0 0 00000000 000001 00 -> R1=84, R2=42, R3=0, R4=0
10/ 1 1 0 0 00000000 010110 00 -> R1=84, R2=0, R3=0, R4=0
11/ 1 1 1 0 11111111 100001 00 -> R1=84, R2=0, R3=12, R4=0
12/ 0 1 0 0 00000000 000010 11 -> R1=88, R2=0, R3=12, R4=0

En hexadécimal ça donne :

1/ C045C	4/ 40004	7/ C2A5C	10/ C0058
2/ 40004	5/ 1FF00	8/ 40004	11/ EFF84
3/ 40004	6/ C0008	9/ 40004	12/ 4000B

Comme j'ai modifié les instructions programmes il me faut modifier le décodeur comme suit :

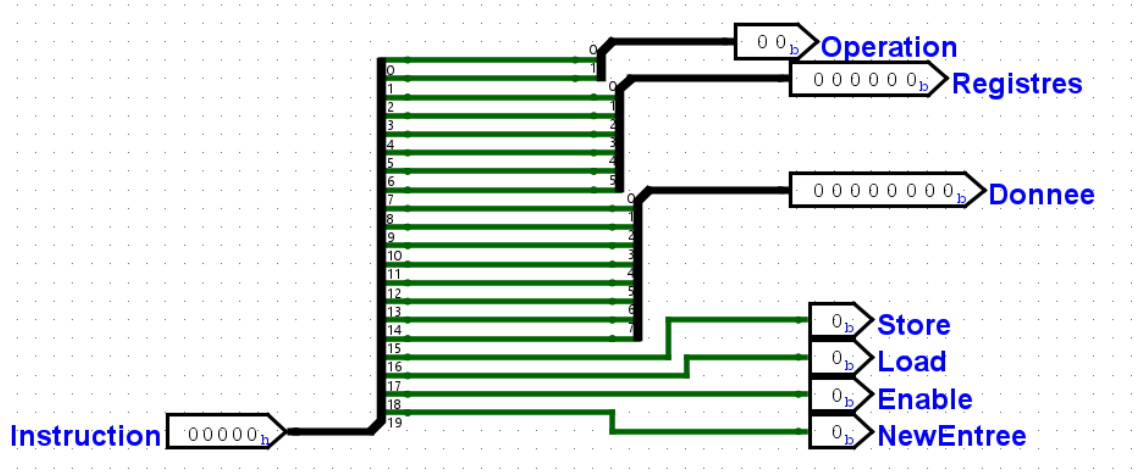

Figure 24 : Décodeur V2

Il me suffit maintenant de relier le tout pour avoir un processeur avec de la mémoire RAM :

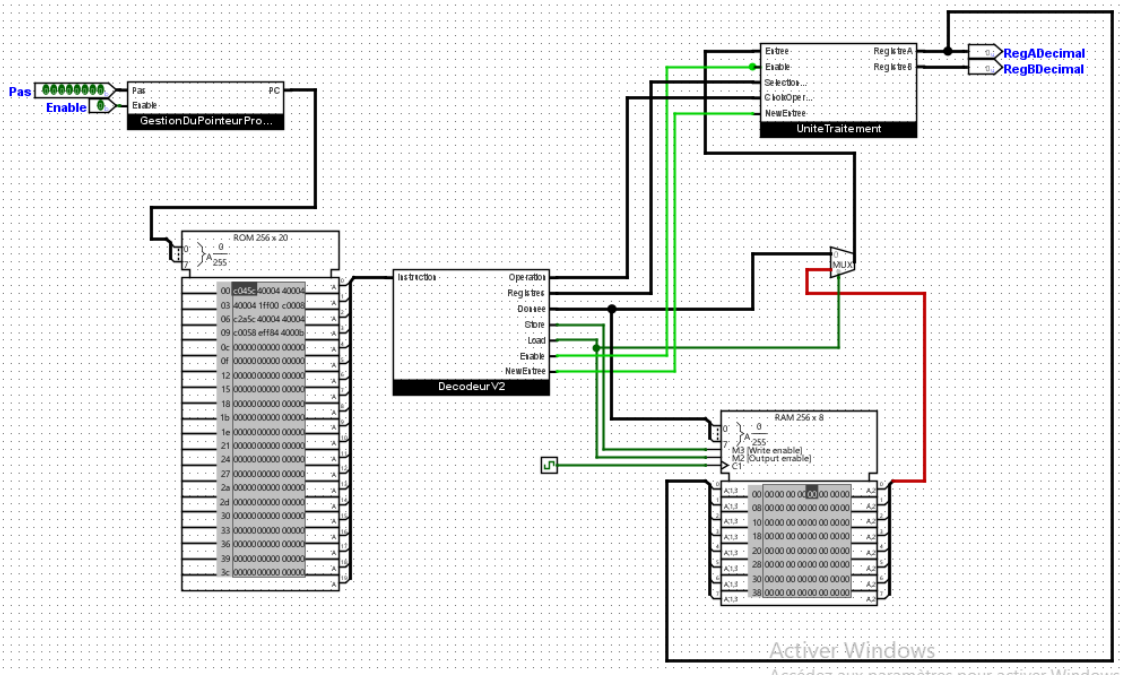

Figure 25 : Processeur avec RAM

9. INSTRUCTION DE BRANCHEMENTS

Pour le moment mon processeur ne peut pas réaliser de branchements (saut d'un endroit du programme vers un autre). Pour palier cela, j'ai besoin d'une nouvelle instruction dans ma mémoire ROM qui mettra à jour le compteur programme du processeur. Mon jump ne passera pas de l'instruction x à y si je lui dis JUMP y mais il sautera de y instructions. Il me faut donc 1 bit pour savoir si je jump ou non. Je me sers des 8 bits pour savoir de combien le compteur programme doit avancer (donc un saut de 255 au maximum). J'ai donc besoin de 21 bits pour mes instructions.

Je définis une instruction tel que :

0 0 0 0 00000000 000000 00 0

- Le premier bit est pour savoir s'il y a une nouvelle entrée (1=oui, 0=non)
- Le deuxième bit est pour le enable donc est ce qu'on a besoin d'écrire (1=oui, 0=non)
- Le troisième bit est pour savoir si on fait un load (1=oui, 0=non)
- Le quatrième bit est pour savoir si on fait un store (1=oui, 0=non)
- Les 8 prochains bits sont les nombres à manipuler

- Les 6 bits d'après sont pour le choix de registre à écrire et ceux à sortir
- Les 2 avant-derniers bits sont pour l'opération à faire
- Le dernier bit sert à savoir s'il y a un jump ou non

Je veux faire l'opération 3*4=12 dont le résultat est stocké dans la RAM, je fais 42+42, je load le résultat d'avant pour le XOR avec celui-ci ce qui donne 88.

1/ Je fais venir 4 que stocke dans R2 (somme avec un registre à 0 (R4))
2/ Je fais une somme entre R2 et R1 (à 0 pour l'instant) et je stocke le résultat dans R1
3/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
4/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
5/ Je store R1 dans la RAM à la dernière adresse
6/ Je remets R1 à 0 en envoyant 0 et en sommant avec un registre à 0 (R3)
7/ Je fais venir 42 que stocke dans R2 (somme avec un registre à 0 (R4))
8/ Je fais une somme entre R2 et R1 (à 0 pour l'instant) et je stocke le résultat dans R1
9/ Je fais une somme entre R2 et R1 et je stocke le résultat dans R1
10/ Je remets R2 à 0 en envoyant 0 et en sommant avec un registre à 0 (R3) (je fais juste attention de toujours avoir un registre à 0 parce que je n'ai pas de reset)
11/ Je load la valeur stockée à la dernière place de la mémoire RAM dans R3
12/ Je fais un jump de 10
23/ Je fais un XOR entre R3 et R1 (à 84 pour l'instant) et je stocke le résultat dans R1

En binaire cela correspond à :

1/ 1 1 0 0 00000100 010111 00 0 ->R1=0, R2=4, R3=0, R4=0
2/ 0 1 0 0 00000000 000001 00 0 ->R1=4, R2=4, R3=0, R4=0
3/ 0 1 0 0 00000000 000001 00 0 ->R1=8, R2=4, R3=0, R4=0
4/ 0 1 0 0 00000000 000001 00 0 ->R1=12, R2=4, R3=0, R4=0
5/ 0 0 0 1 11111111 000000 00 0 ->R1=12, R2=4, R3=0, R4=0
6/ 1 1 0 0 00000000 000010 00 0 ->R1=0, R2=4, R3=0, R4=0
7/ 1 1 0 0 00101010 010111 00 0 ->R1=0, R2=42, R3=0, R4=0
8/ 0 1 0 0 00000000 000001 00 0 ->R1=42, R2=42, R3=0, R4=0
9/ 0 1 0 0 00000000 000001 00 0 ->R1=84, R2=42, R3=0, R4=0
10/ 1 1 0 0 00000000 010110 00 0 ->R1=84, R2=0, R3=0, R4=0
11/ 1 1 1 0 11111111 100001 00 0 ->R1=84, R2=0, R3=12, R4=0
12/ 0 0 0 0 00001010 000000 00 1 ->R1=84, R2=0, R3=12, R4=0
13 à 21/ que des 0 on ne fait rien (on pourrait faire autre chose mais ça ne serait pas pris en compte dans tous les cas)
22/ 0 1 0 0 00000000 000010 11 0->R1=88, R2=0, R3=12, R4=0

En hexadécimal ça donne :

1/ 1808B8	5/ 3FE00	9/ 80008	22/ 80016
2/ 80008	6/ 180010	10/ 1800B0	
3/ 80008	7/ 1854B8	11/ 1DFF08	
4/ 80008	8/ 80008	12/ 1401	

Comme j'ai modifié les instructions programmes il me faut modifier le décodeur comme suit :

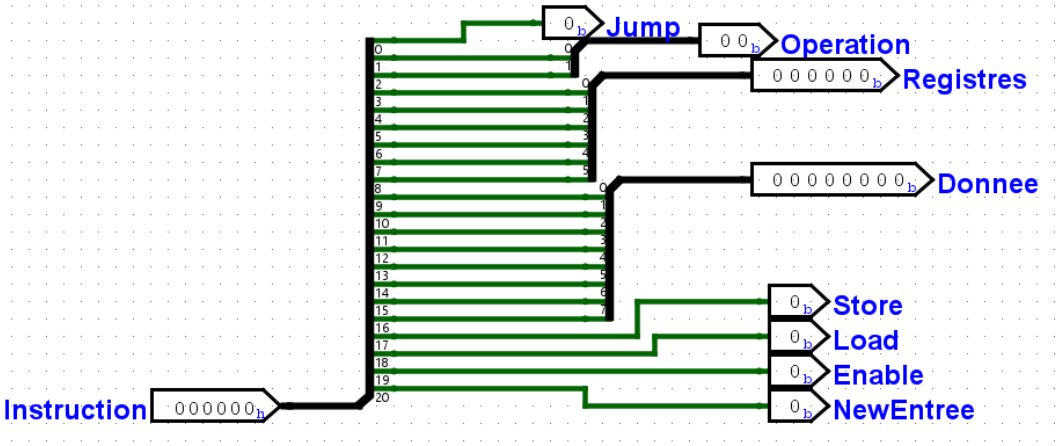

Figure 26 : Décodeur V3

Il me suffit maintenant de relier le tout pour avoir un processeur avec des branchements possible :

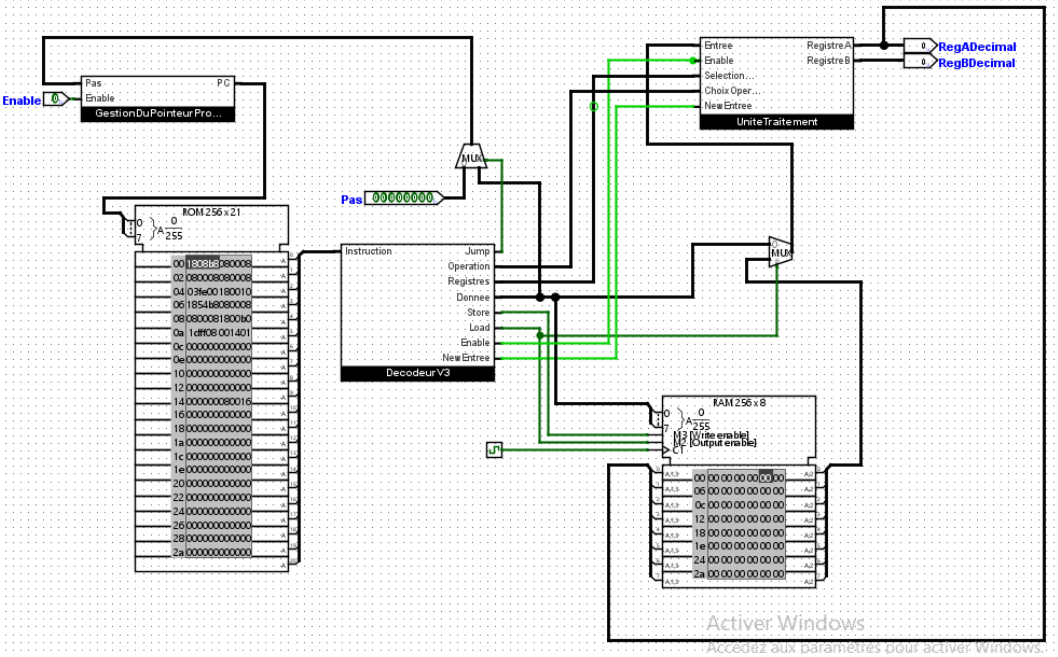

Figure 27 : Processeur final avec branchements

III. CONCLUSION

C'est avec l'ajout de cette instruction de branchement que s'achève ce projet dans la matière "Langage d'assemblage et microprocesseur". Ce projet m'a permis de mieux assimiler plusieurs points liés à l'hardware. J'avais déjà utilisé un logiciel similaire à Logisim-evolution lors de mes études mais ce fût une découverte des modules ROM et RAM. Même si le logiciel rend les étapes instantanées au vu des branchements on se rend encore mieux compte de la nécessité d'une optimisation et d'une grande quantité de registre pour l'unité de traitement. En effet, on se doute que dans la réalité un load/store demande énormément de temps ce qui fait un goulot d'étranglement pour un processeur. J'ai également mieux assimilé la méthode d'extraction d'équations d'une table de vérité. Si je devais refaire ce projet je prendrais plus de temps pour faire d'autres portes tel que NAND ou NOR mais encore la réalisation d'un jump différent celui de passer d'une instruction X à une instruction Y de la ROM. Ce fût un projet très prenant et intéressant à mon sens et à le mérite d'exister.