Thomas Dagier 19.06.2020

# Projet SLO : processeur et mémoire de données, Rapport de fin de projet

h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Structure du rapport :

- I/ Présentation du projet et objectifs de réalisation
- II/ Démarche et évolution chronologique
- III/ Description du travail assembleur
- IV/ Conclusion

## I/ Présentation du projet et objectifs de réalisation

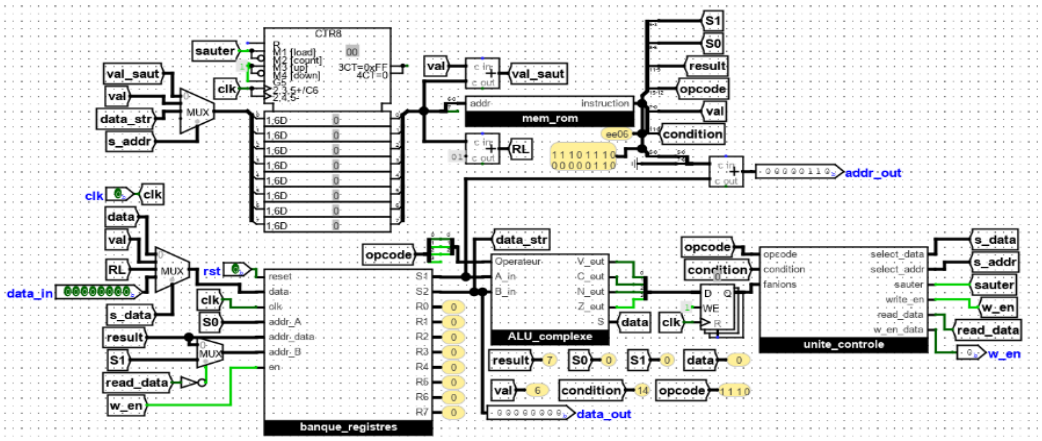
Le projet de SLO se compose d'une partie exercices et d'une partie libre. L'objectif principal est de faire une application en utilisant le processeur crée pendant les séances d'exercice du second semestre. Les exercices sur un total de 7 semaines nous ont donc permis de faire un processeur simple et utilisable pour toute sorte d'application. Le cahier des charges fournis des informations sur l'application.

Mon premier objectif était de faire un jeu où un pixel, soumis à la gravité, pourrait sauter pour esquiver des obstacles générés aléatoirement. Pour donner suite aux tests que j'ai pu faire durant la 6eme et 7eme semaine du projet, j'ai fait le choix de faire un jeu plus simple compte tenu du temps que nous avions à disposition et des autres projets que nous avions à rendre.

Le projet, dans l'état actuel, est un mini-jeu simple dans lequel un joueur peut utiliser deux boutons pour contrôler un pixel. Ce dernier, qui représente le joueur, doit esquiver les obstacles qui apparaissent de manière pseudo-aléatoire. Si le joueur passe l'obstacle, un nouveau apparait mais si le joueur échoue, le jeu recommence.

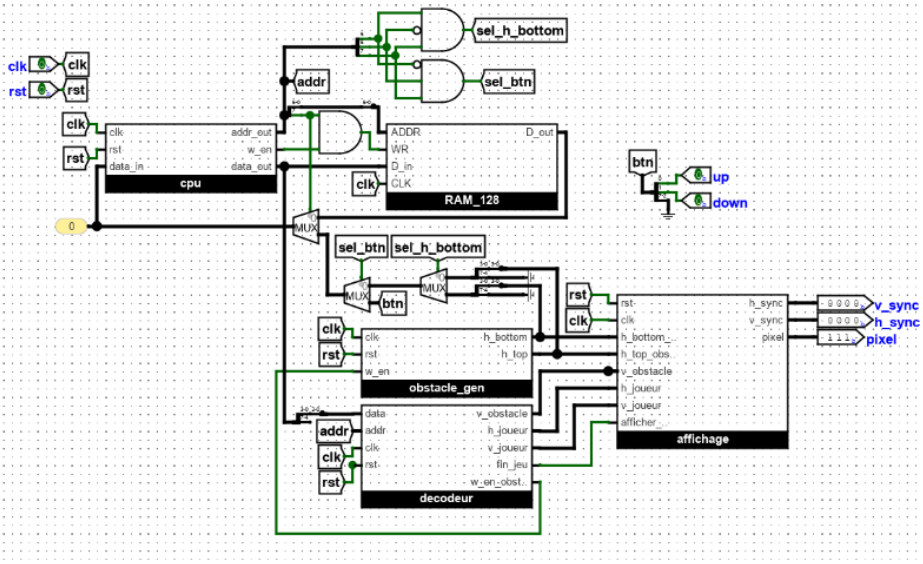
## II/ Démarche et évolution chronologique

Au niveau de l'architecture du projet, l'élément principal est le CPU. Il est composé d'une banque de 8 registres, d'une ALU complexe 8 bits , d'une unité de contrôle et d'une mémoire d'instructions (8 bits d'adresse).



Ce CPU est directement connecté à une mémoire RAM (128 bits) et à des périphériques. Ces composants sont accessibles par une même adresse (addr_out). Il y a deux périphériques d'entrée, un bus de deux boutons et un générateur pseudo-aléatoire (fournis par logisim). En sortie, le périphérique utilisé est un RGB vidéo : un écran de 16x16 pixels.

La première grande partie du projet est de réaliser les exercices qui seront nécessaires pour le bon fonctionnement du CPU. Les parties importantes sont la connexion entre la banque de registres et l' ALU complexe (exercice 2), l'affectation de valeur, les sauts conditionnels et inconditionnels (exercice 3), les Branch and Link et Branch Register (exercice 4). Pour connecter le CPU à la RAM et aux périphériques, on utilise les Load et Store (exercice 5). Si l'adresse est inférieure à 128, on accède aux données de la RAM, sinon on accède aux données des périphériques, toujours en lecture ou écriture.




Une fois le processeur réalisé et les accès aux périphériques installés, il faut désormais prévoir les périphériques à utiliser. Le premier périphérique, le plus évident est un bus de deux boutons. Ce dernier est adressable à l'adresse 0xC8. On utilise aussi un générateur d'obstacle. Ce dernier retourne deux positions qui permettrons de créer un obstacle. Une première est lue à l'adresse 0x80 et une deuxième à l'adresse 0xA0.

Ces données seront manipulées dans les instructions qui créeront le jeu et sont également envoyées dans le périphérique de sortie. L'écran fonctionne de la même manière qu'un GPU. Ce composant a besoin de lire toutes les informations de positions (du joueur et de l'obstacle). Son rôle est simplement de colorer les pixels d'une couleur en fonction des coordonnées lues et modifiées par le CPU. Il n'y a aucune modification des valeurs ou calculs qui sont faits dans le GPU (bloc affichage du code).

## III/ Description du travail assembleur

Lorsque tous les composants ont été testés ensembles ou indépendamment en utilisant des constantes logisim, il a fallu commencer à créer le code assembleur. La démarche nous a été un peu simplifiée grâce au compiler logisim fournit.

```
bl [r7] initialisation
bl [r7] updateAffichage
bl [r7] setData
bl [r7] detecterColision
bl [r7] renouvelerObstacle
b -4

initialisation:
r0 = 7
r1 = 4
r5 = 160
ld r2,0[r5]
r5 = 128
ld r3,0[r5]
r4 = 15
br [r7]

updateAffichage:
r6 = 1
r4 = r4 - r6
r6 = 200
ld r5,0[r6]
r6 = 1
r6 = r6 - r5
bcz 6
r6 = 2
r6 = r6 - r5
bcz 7
br[r7]
r6 = 1
r6 = r0 - r6
r0 = r6
br [r7]
r6 = 1
```

J'ai fait le choix de diviser le code en plusieurs sous-fonctions pour satisfaire plusieurs objectifs.

Le premier est de clairement identifier ce-dont on a besoin et comment le faire avec le moins d'instructions possibles.

Le second est de clairement séparer la gestion de l'affichage des calculs et des détections de collisions.

Enfin, séparer les fonctions permet de tester indépendamment chaque partie pour vérifier le bon fonctionnement plutôt que de tout débugger à la fin.

Une partie importante du projet qui m'a fait gagner beaucoup d'instructions est le fait de stocker les valeurs les plus importantes dans des registres et ne jamais écrire par-dessus. Cela permet d'éviter de lire et réécrire dans la mémoire régulièrement quand ce n'est pas nécessaire.

Le gain d'instructions est un procédé fondamental pour le projet. En effet, moins il y a de coup de clock et plus on pourra détecter les changements au niveau des boutons et de l'affichage. Cela permet d'être plus précis dans la gestion de l'affichage et donc du jeu.

L'implémentation du code assembleur a nécessité de modifier un peu le compteur d'instructions et les calculs à faire dans le cas du BL et BR. Cet ajout a surtout permis d'identifier des erreurs dans le câblage du projet.

En testant l'assembleur, j'ai aussi découvert des fonctionnalités qui étaient à modifier ou tout simplement à ajouter dans le code. Par exemple la gestion du personnage s'il cogne le bord supérieur ou inférieur du jeu.

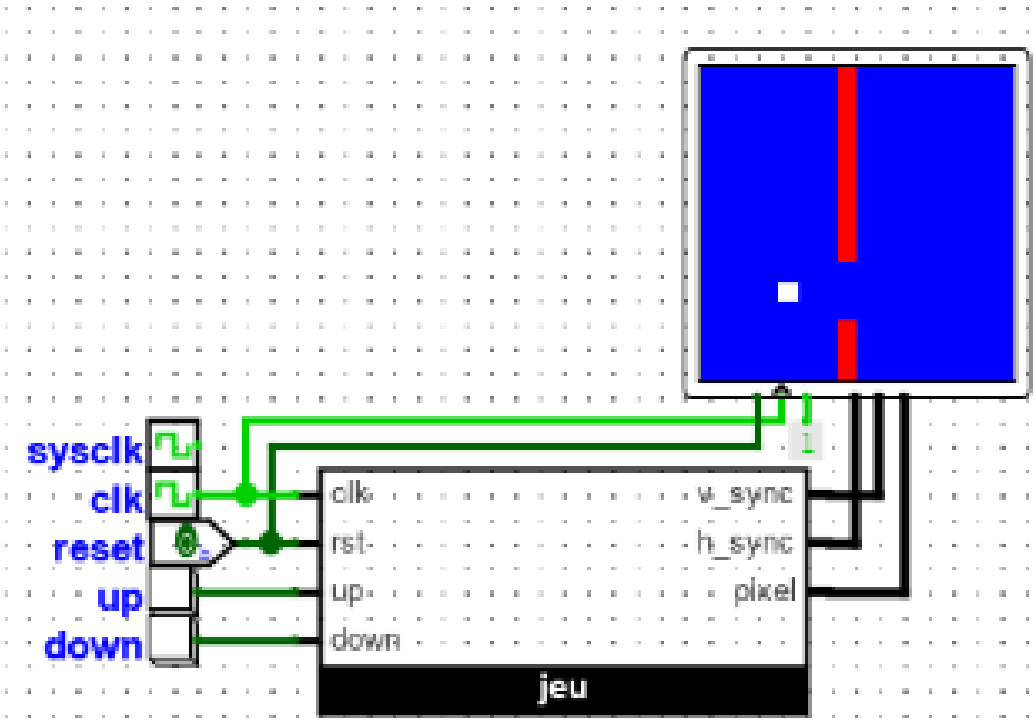
Pour modifier les valeurs qui ne vont pas dans un registre, on utilise un décodeur. Ce dernier permet d'utiliser les résultats du CPU dans le reste du jeu comme générer un nouvel obstacle ou terminer le jeu à un moment précis déterminé par le CPU.

## I V/ Conclusion

Ce projet a permis de vraiment bien comprendre l'architecture d'un ordinateur. En évoquant les différents composants en cours, le projet nous a permis de mieux comprendre et surtout de mettre en pratique les notions évoquées. Le sujet et les exercices étaient très instructifs et nous ont permis d'approfondir nos connaissances et d'apprendre les bases du code assembleur.

Il est vrai qu'en travaillant sur le cahier des charges, je n'avais pas vraiment conscience des capacités de notre CPU. Ceci m'a conduit à réaliser un projet que je trouve assez complexe et je reste un peu frustré de ne pas avoir pu y consacrer plus de temps pour un projet que j'ai trouvé intéressant.

Je reste très content du résultat au vu du temps passé, notamment car j'ai su utiliser le RGB vidéo et un CPU construit à partir de rien qui semble très bien fonctionner et j'ai également appris beaucoup de notions par la pratique qui complètent la théorie de cette première année.




Une perspective d'amélioration possible serait d'afficher le score de la partie. Il faudrait pour cela écrire et lire dans la mémoire avant de relancer un nouvel obstacle.

Concernant Logisim plus spécifiquement, j'ai le sentiment que mon ordinateur bride les hautes fréquences pour la clock car le programme fonctionne parfaitement sur l'ordinateur de mon frère alors que l'ordinateur a déjà 7 ans. De ce fait il y a quelques problèmes dans l'affichage et les cliques des boutons qui sont spécifiques à mon ordinateur (peut-être à cause de Java ?).