EIDD - Architecture micro-processeurs

Architecture Processeur LC-3

Auteurs :
M^{lle} MIMOUNA Nour Elkamar

Encadrants :
M. Piriou Pierre-Yves
M. Férée Hugo

Version du
12 janvier 2025

Table des matières

1 Introduction

2 Instructions LC-3 à implémenter

2.1 Decode-IR

Le Decode-IR dans l'architecture LC-3 est l'étape où le processeur analyse l'instruction stockée dans le registre IR pour déterminer l'opération à effectuer. Cela inclut l'extraction de l'op-code et des opérandes pour identifier le type d'instruction et les actions correspondantes à entreprendre.Une fois l'instruction chargée dans le registre IR, les différentes parties du circuit peuvent accéder à ses champs (opcode, opérandes, etc.) afin de déterminer les opérations à réaliser.

En effet, l'op-code, et plus spécifiquement les bits 12 et 13, permettent d'identifier le type d'instruction à exécuter.

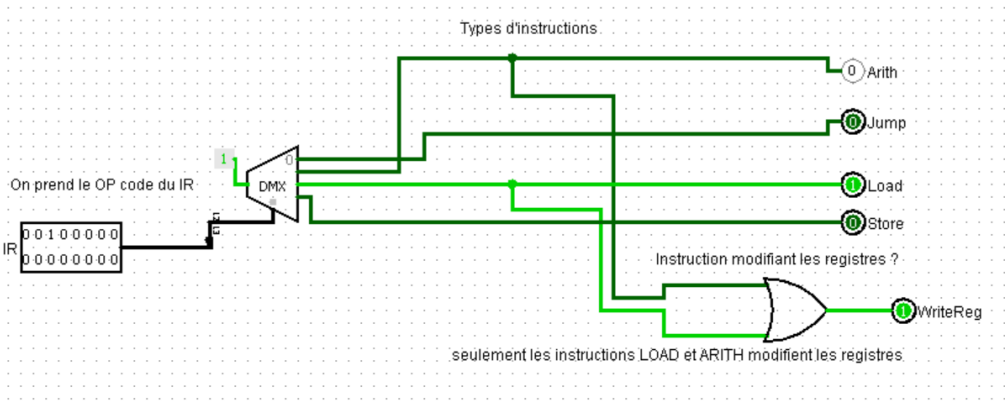


FIGURE 1 – Le circuit DecodeIR.

2.1.1 Instructions Arithmétiques

Pour déterminer la fonction logique de $f(IR) = \text{Arith}$, nous avons d'abord analysé les op-codes des instructions correspondantes. En utilisant un tableau de Karnaugh, nous avons constaté qu'il est possible d'ignorer les bits 15 et 14. Ainsi, la fonction dépend uniquement des bits 12 et 13, et peut être exprimée sous la forme :

$$f = a'b$$

où a représente le bit 13 et b le bit 12. Cela correspond à la sortie 1 du démultiplexeur.

			Op-code				Arguments										
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT DR,SR	DR ← not SR	*	1	0	0	1	DR			SR			1	1	1	1	1	1
ADD DR,SR1,SR2	DR ← SR1 + SR2	*	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5				
AND DR,SR1,SR2	DR ← SR1 and SR2	*	0	1	0	1	DR			SR1			0	0	0	SR2		
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5				

FIGURE 2 – Les instructions Arithmétiques.

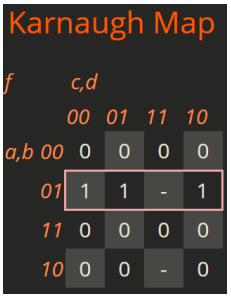


FIGURE 3 – Tableau de Karnaugh des instructions Arithmétiques.

2.1.2 Instructions JUMP

On applique la même méthode pour déterminer la fonction logique des instructions Jump. Cette fois-ci, nous constatons qu'il est également possible d'ignorer les bits 15 et 14. La fonction est alors donnée par :

$$f = a'b'$$

où a représente le bit 13 et b le bit 12. Cela correspond à la sortie 0 du démultiplexeur.

2.1.3 Instructions LOAD

On applique la même méthode pour déterminer la fonction logique des instructions Load. La fonction est donnée par :

$$f = ab'$$

où a représente le bit 13 et b le bit 12. Cela correspond à la sortie 2 du démultiplexeur.

2.1.4 Instructions STORE

On applique la même méthode pour déterminer la fonction logique des instructions STORE. La fonction est donnée par :

$$f = ab$$

où a représente le bit 13 et b le bit 12. Cela correspond à la sortie 3 du démultiplexeur.

2.1.5 WriteReg

En effet, cette sortie indique si l'instruction en cours (IR) modifie les registres, autrement dit si elle modifie les flags NZP. D'après le tableau "Récapitulatif des Instructions", on constate que seules les instructions LOAD et ARITH modifient les registres. Ainsi, la fonction logique de `WriteReg` est donnée par :

$$\text{WriteReg} = \text{LOAD ou ARITH.}$$

2.2 NZP

Dans l'architecture du LC-3, le circuit `NZP` est responsable de la gestion des indicateurs d'état (ou flags), qui permettent au processeur de déterminer la nature du dernier résultat produit par une opération. Voici ses rôles principaux :

— **N (Negative)** : Le dernier résultat est négatif.
— **Z (Zero)** : Le dernier résultat est nul (égal à 0).
— **P (Positive)** : Le dernier résultat est strictement positif.

Pour déterminer le signe du registre `RES`, on utilise un comparateur qui compare `RES` avec 0, et un splitter qui sépare les 3 bits du résultat du comparateur.

En effet, le résultat ne s'affiche que si `WriteReg` est égal à 1 ; sinon, cette valeur n'est pas mise à jour. Ainsi, on constate l'importance d'une composante qui mémorise la valeur antérieure de `NZP`.

Décision de contrôle : Les indicateurs `NZP` sont utilisés dans les instructions de saut conditionnel (`BR` - Branch). Cela correspond aux bits 9, 10 et 11 du `BR`. Le test de saut est vrai si l'un des flags est vrai.

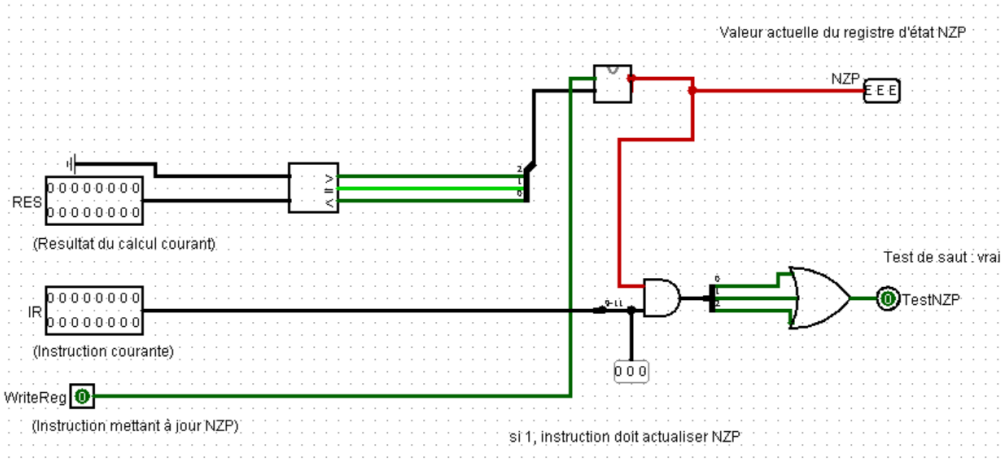


FIGURE 4 – Le circuit NZP.

2.2.1 Verrou 3 bit

Dans le circuit `NZP` du LC-3, le verrou (ou latch) à 3 bits joue un rôle crucial pour maintenir les valeurs des indicateurs d'état (`N`, `Z`, `P`) jusqu'à ce qu'elles soient mises à jour par une instruction.

Fonctionnement global :

— **Signal d'entrée** : Les valeurs des indicateurs (`N`, `Z`, `P`) sont calculées par le processeur en fonction du dernier résultat.

— **Mise à jour conditionnelle** : Lorsque le signal de contrôle est activé, les verrous mettent à jour leurs valeurs avec les entrées respectives.
— **Conservation des données** : Si le signal de contrôle est désactivé, les verrous conservent leurs valeurs précédentes, maintenant ainsi la cohérence des indicateurs.

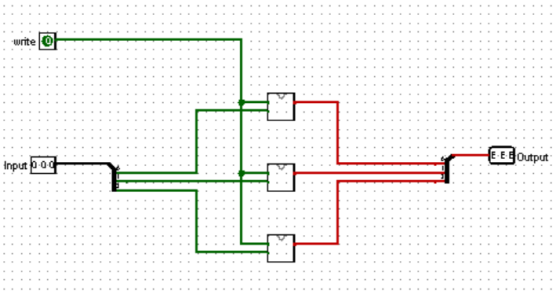


FIGURE 5 – Le circuit Verrou 3 Bit.

2.2.2 Verrou 1 bit

Pour créer un verrou 3 bits dans le circuit NZP, on peux utiliser trois verrous 1 bit indépendants, chacun étant responsable de stocker un des indicateurs (N, Z, ou P).

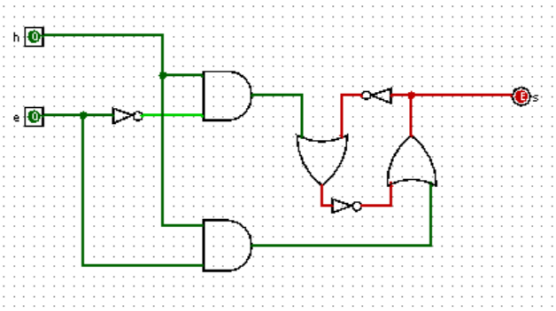


FIGURE 6 – Le circuit Verrou 1 Bit.

2.3 WriteVal

Le circuit `WriteVal` dans l'architecture du LC-3 joue un rôle clé dans la gestion de l'écriture des valeurs dans les registres ou la mémoire, en fonction des besoins de l'instruction en cours. Nous avons été contraintes d'utiliser un multiplexeur (`MUX`) à 8 entrées pour prendre en compte la condition selon laquelle, si l'instruction est `Load`, alors `MemOUT` est sélectionné sans tenir compte des autres entrées.

La fonction de `RegIN` peut être décrite comme suit :

$$\begin{aligned}\text{RegIN} = &(\text{JSR} \cdot \text{PC}) \\ &+ (\text{LEA} \cdot (\text{PC} + \text{SEXT(Offset9)})) \\ &+ (\text{Load} \cdot \text{MemOUT}) \\ &+ (\neg\text{JSR} \cdot \neg\text{LEA} \cdot \neg\text{Load} \cdot \text{ALURes})\end{aligned}$$

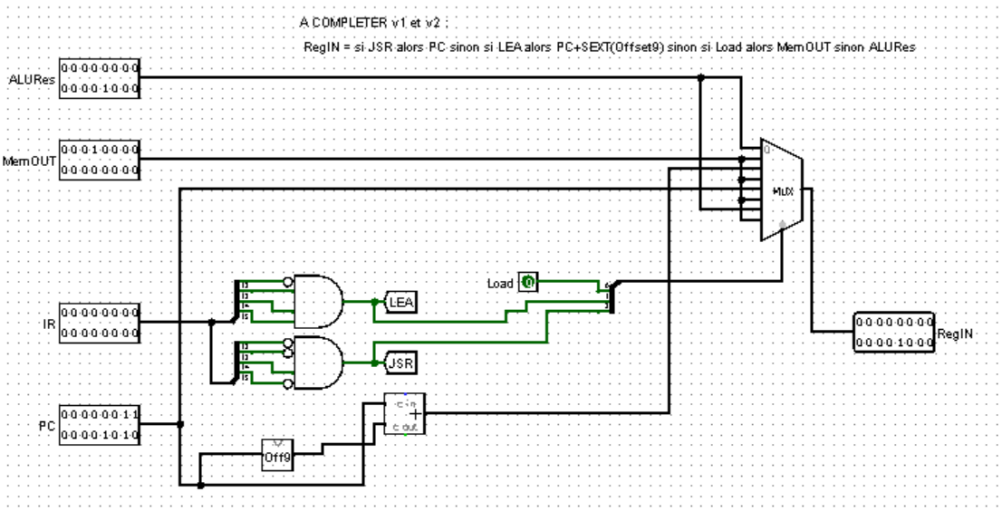
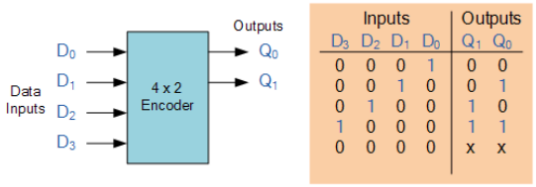


FIGURE 7 – Le circuit WriteVal

3 Instruction bit scan

L'instruction Bit Scan est utilisée pour localiser le premier bit à 1 (ou parfois le premier bit à 0, selon la variante de l'instruction) dans un registre ou une donnée binaire, en commençant par le bit de poids faible (LSB, Least Significant Bit).

Pour implementer cette instruction on utilise Un priority encoder (codeur de priorité) est un circuit combinatoire utilisé pour détecter le bit actif ayant la plus haute priorité dans une entrée binaire, et il encode sa position en sortie sous forme binaire.



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

FIGURE 8 – Priority Encoder 4 bits

3.1 BSB - bit scan backward

Un priority encoder 16 bits pour l'instruction BSB doit être configuré pour renvoyer la position du premier bit à 1 en partant du poids fort.

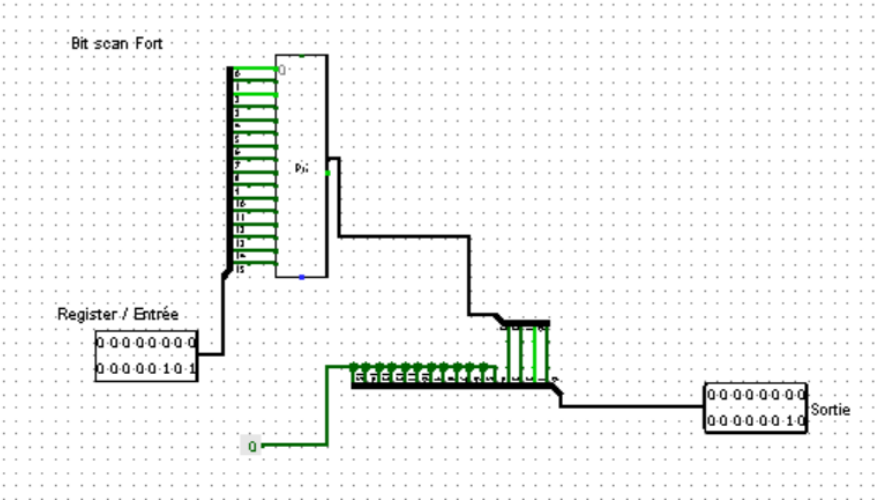


FIGURE 9 – Circuit BSB

3.2 BSF

L'instruction BSF recherche le premier bit actif à partir du poids faible (LSB). Cela signifie qu'elle commence par analyser le bit 0 (le plus à droite) et cherche le premier bit égal à 1.

Inversion des entrées du priority encoder

Dans le cas où l'on souhaite inverser l'ordre des bits pour faciliter l'utilisation du priority encoder classique (qui fonctionne normalement de LSB vers MSB), les entrées sont inversées. Le bit 0 est placé sur l'entrée 15, et ainsi de suite, de manière à ce que le bit de poids faible soit en haut et le bit de poids fort en bas. Cela permet au priority encoder de renvoyer la position correcte du premier bit à 1, tout en respectant l'ordre attendu par l'instruction BSF, qui scanne à partir du poids faible.

Cas où l'entrée est nulle (tous les bits à 0)

Lorsque l'entrée est 0 (c'est-à-dire que tous les bits sont à 0), le priority encoder peut renvoyer une sortie invalide (souvent `XXX` ou une valeur non définie), car il n'y a pas de bit actif à 1. On vérifie ceci avec le north pin du priority encoder.

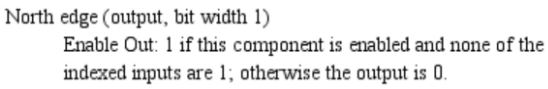


FIGURE 10 – Le guide du logisim

a) Vérification avant la soustraction

Dans ce cas, il est nécessaire de vérifier si l'entrée est 0 avant de procéder à la soustraction. Si l'entrée est 0, la valeur du BSB est renvoyé car le BSF de 0 est egal au BSB de 0.

b) Cas où l'entrée contient un bit actif

Si l'entrée contient au moins un bit égal à 1, la soustraction est effectuée normalement.

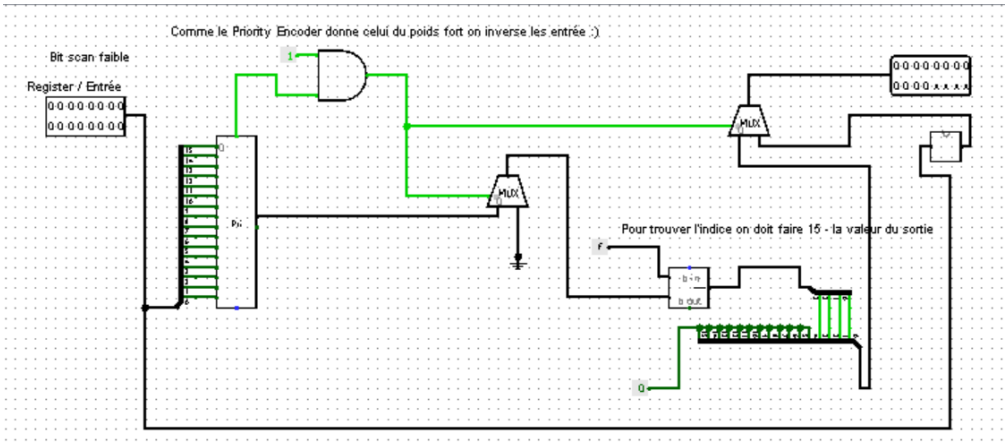


FIGURE 11 – Circuit BSF

3.3 ALU

L'ALU (Arithmetic Logic Unit) est responsable du traitement des opérations arithmétiques et logiques dans le processeur LC3. Elle effectue des opérations comme l'addition, la soustraction, et des opérations logiques comme le ET (AND), le NON (NOT), ainsi que des instructions de type `bit scan`.

L'opcode (code opérationnel) d'une instruction LC3 détermine quelle opération sera effectuée par l'ALU. L'ALU est contrôlée par les bits de l'opcode, notamment les bits `E2` et `E1`, qui permettent de sélectionner respectivement l'instruction et le type d'opération.

Le bit E2 (bits 12 et 13 de l'opcode)

Les bits `E2` (qui correspondent aux bits 12 et 13 de l'opcode) permettent de sélectionner l'instruction à exécuter. Selon la valeur de `E2`, l'ALU effectue l'une des opérations suivantes :

- `00` : Addition (`ADD`)
- `01` : ET logique (`AND`)
- `10` : NON logique (`NOT`)
- `11` : Bit scan (`BSF` ou `BSB`)

Le bit E1 (bit 5 de l'opcode)

Le bit `E1` (qui correspond au bit 5 de l'opcode) permet de déterminer le type d'opération au sein des instructions `ADD` et `AND`. Ce bit distingue les différentes

versions de ces instructions, en fonction des registres ou des constantes utilisées :

— Si `E1 = 0`, l'opération se fait entre deux registres (`REG1`, `REG2`).
— Si `E1 = 1`, l'opération se fait entre un registre (`REG1`) et une constante (`CNST`).
— Pour les instructions `bit scan`, `E1` peut déterminer si l'opération est effectuée sur un bit spécifique dans les bits de condition (`BFB` ou `BFF`).

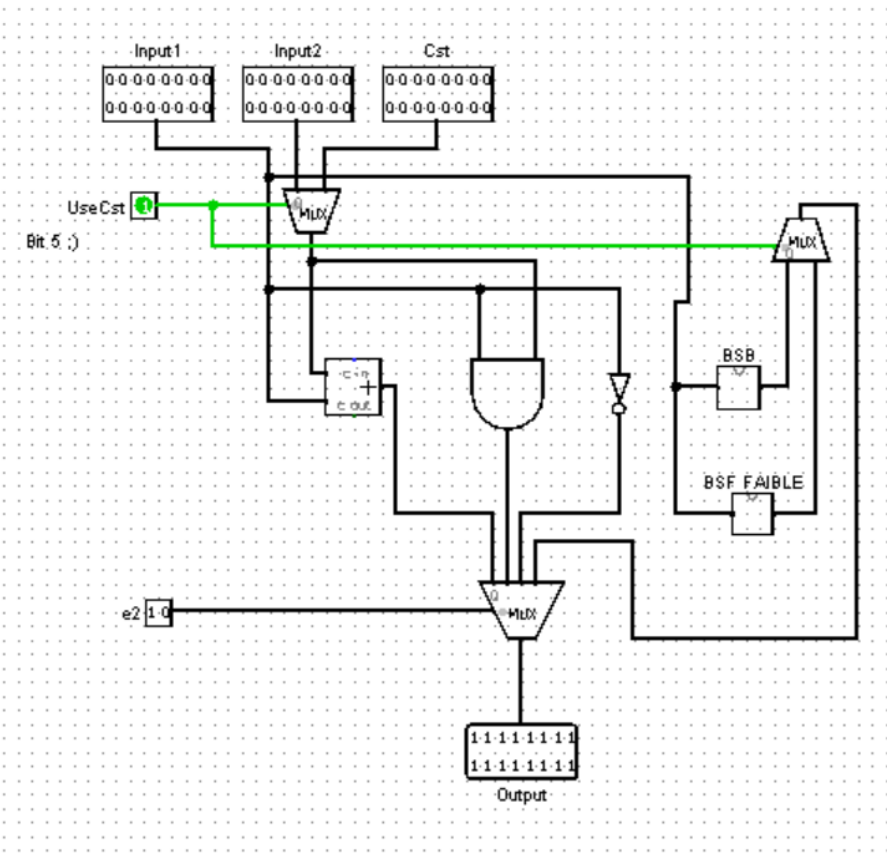


FIGURE 12 – Circuit ALU

4 Programmation en LC-3 étendu

Dans ce projet, nous avons également développé des routines en langage d'assemblage pour gérer les occurrences, la comparaison de registres, la manipulation binaire, les débordements et la multiplication saturée. Certaines étaient simples, d'autres ont nécessité des approches plus complexes pour des résultats optimaux.

4.1 Longueur, Première occurence, Dernière occurence, Comparaison, Copie, Copie de taille fixée

Les six premières routines étaient simples, utilisant des instructions de base étudiées en cours. Par exemple, pour les occurrences, nous avons utilisé des pointeurs et des instructions comme AND et ADD. La comparaison de registres a été faite avec AND et BRz. La copie de valeurs entre registres ou zones de mémoire était directe, et la copie de taille fixée a utilisé un traitement itératif. Ces étapes ont permis de bien comprendre les concepts de base du langage d'assemblage.

4.2 Poids de Hamming

La routine poids de Hamming a été plus complexe, nécessitant un comptage des bits à 1 dans un registre. Nous avons utilisé des opérations AND et des décalages de bits pour isoler chaque bit et les compter, ce qui a demandé plus de temps et d'attention.

4.3 Prédiction de débordement

La routine de prédiction de débordement a impliqué deux sous-routines : l'extraction du bit de poids fort pour identifier un risque de débordement, et la comparaison de ces bits entre deux registres. Après des tests unitaires, elles ont été combinées pour créer la routine finale, essentielle pour prévenir les débordements lors des opérations sur des registres à largeur fixe.

4.4 Produit saturé 16 bits

La routine de produit saturé 16 bits a été la plus complexe. Elle visait à multiplier deux nombres sans dépasser la capacité de 16 bits. Malgré plusieurs stratégies explorées pour simuler la saturation, une solution totalement robuste

n'a pas été trouvée. Cette partie a toutefois permis de mieux comprendre la gestion des registres à taille fixe et les limites de l'architecture processeur.

5 Conclusion

En conclusion, ce projet a permis de mieux comprendre les mécanismes de l'architecture LC-3, notamment le fonctionnement des circuits de décodage, de gestion des registres et des indicateurs NZP. Cependant, plusieurs difficultés ont été rencontrées, notamment dans la conception des fonctions logiques des différents composants (MUX, verrous) et leur intégration dans un système cohérent. La gestion des signaux de contrôle pour la mise à jour des registres et des indicateurs, ainsi que la gestion correcte des conditions d'exécution des instructions, ont été des défis importants. Malgré ces obstacles, le projet a permis de renforcer notre compréhension du rôle central du contrôle dans l'architecture d'un processeur.