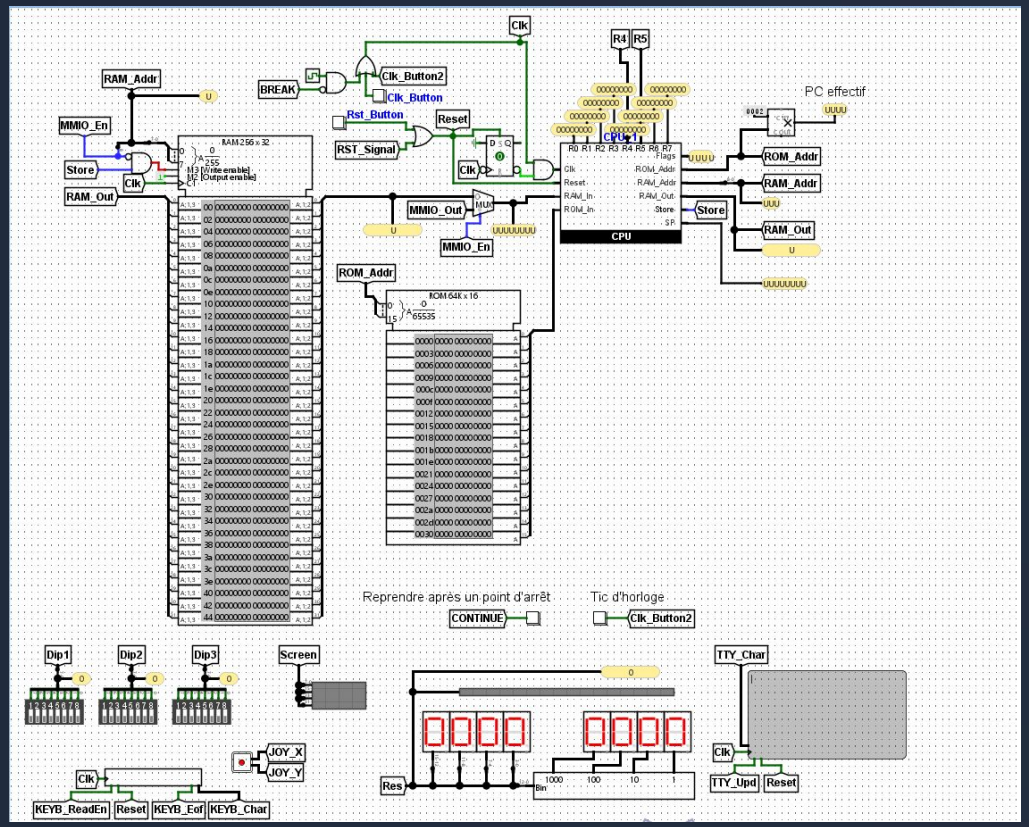
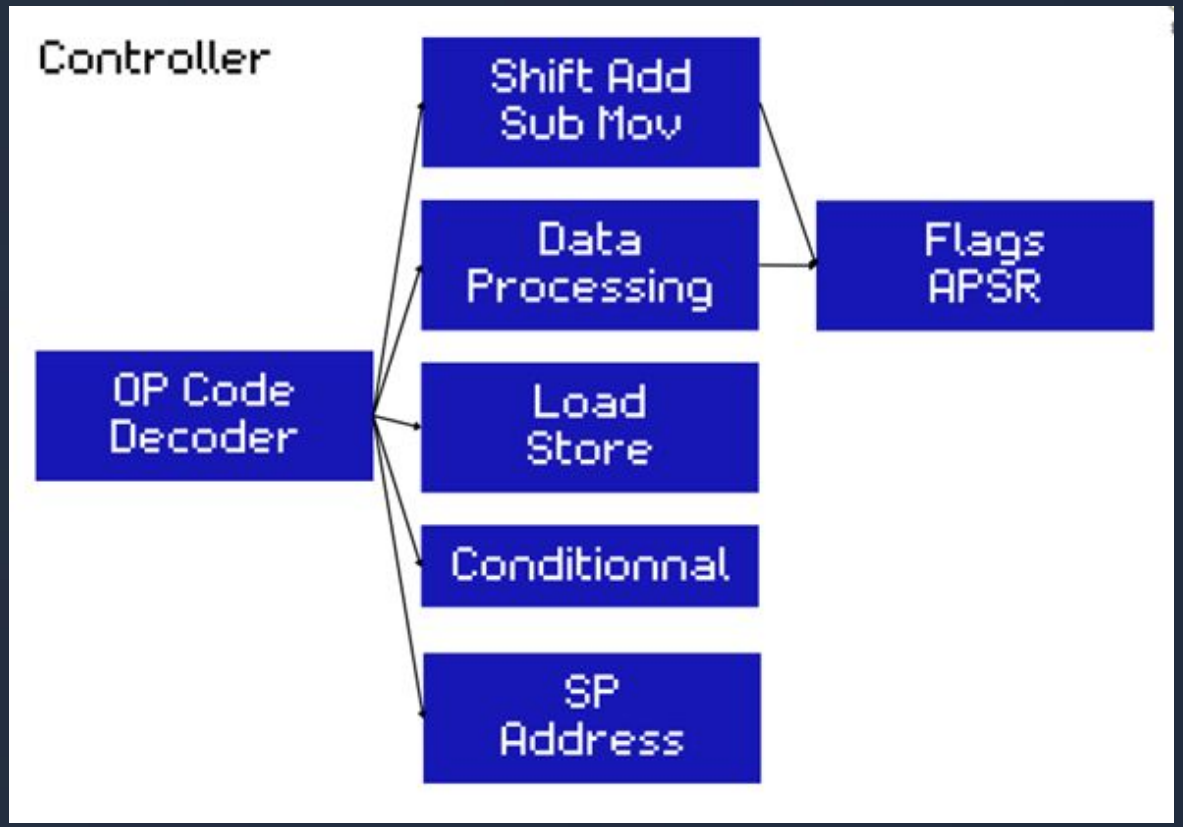
Projet PARM

Anas El Baali
Ilias El hadi
Florian Tamehmacht
El Bah Mohamed Abderrahmane

Introduction et Objectifs






PC
Instruction | Mémoire
Instruction
OP Code Decoder
Shift Add Sub |
Data Process
Load / Store
Conditional
Mov
(ALU_Opcode
Flags_Mask)
Register Bank |
Résultat
Arithmetic Logic
Flags

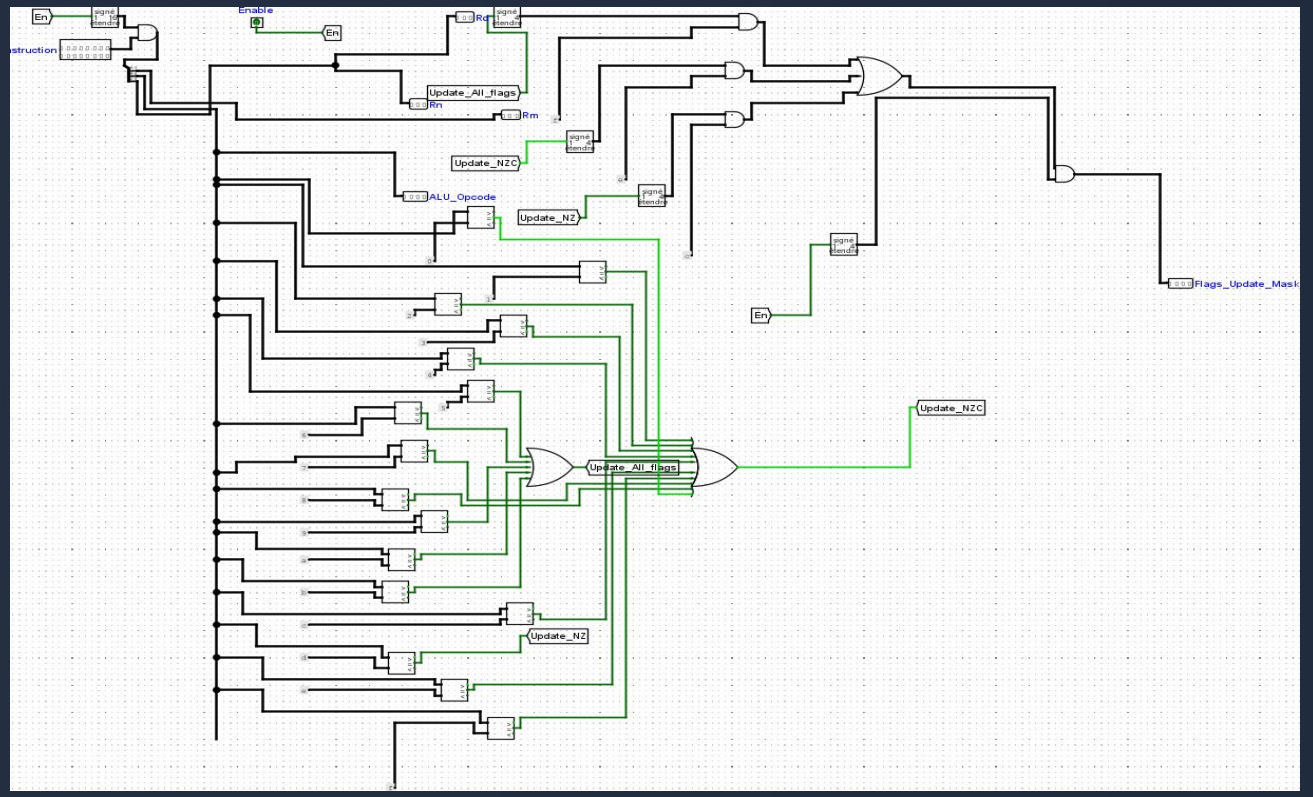
Data Processing

Prépare les opérandes destinées à l’ALU en sélectionnant les sources de données, en appliquant les décalages et en configurant le chemin de données avant l’exécution.

Gérer les instructions de traitement de données

Sélectionner :

- Registre ou immédiat
- Chemins de données vers l’ALU



Flux de fonctionnement (Comment ça circule ?)

1. La Machine récupère une instruction.
2. L'Opcode Decodeur dit : "C'est une addition".
3. Shift_Add_Sub_Mov prépare le ALU_Opcode (ex: 5) et le Flags_Update_Mask.
4. Le Banc Register sort les valeurs des deux registres demandés.
5. L'ALU fait l'addition.
6. Le résultat retourne dans le Banc Register et les Flags sont mis à jour.
7. Si c'est un saut, le bloc Conditional décide de la suite selon les drapeaux.

Conditional

Analyse les drapeaux de l'APSR et décide, selon la condition de l'instruction, si l'exécution continue normalement ou si un saut est effectué.

- **Gérer l'exécution conditionnelle**

Fonctionnement :

- **Lit les flags APSR**
- **Compare avec la condition demandée (EQ, NE, LT, GE…)**
- **Décide :**
  - **Saut pris → modification du PC**
  - **Sinon → exécution normale**

Unité de Décodage et de Contrôle : Shift, Add, Sub & Move

Ce circuit constitue l'interface logique entre l'instruction brute et l'unité de calcul. Il transforme un code machine de 16 bits en commandes synchronisées pour piloter l'ALU et le banc de registres.

I. Décodage et Routage des Opérations

- Logique de sélection : Utilisation des bits 14 et 15 (via Splitter et porte AND) pour piloter un démultiplexeur activant cible.
- Génération d'Opcode : Conversion de chaque instruction en une constante numérique (ex: ADD=5, SUB=6) transmise via un bus OR vers le tunnel ALU_Opcode.l'instruction

Gestion Sélective des Drapeaux (Flags)

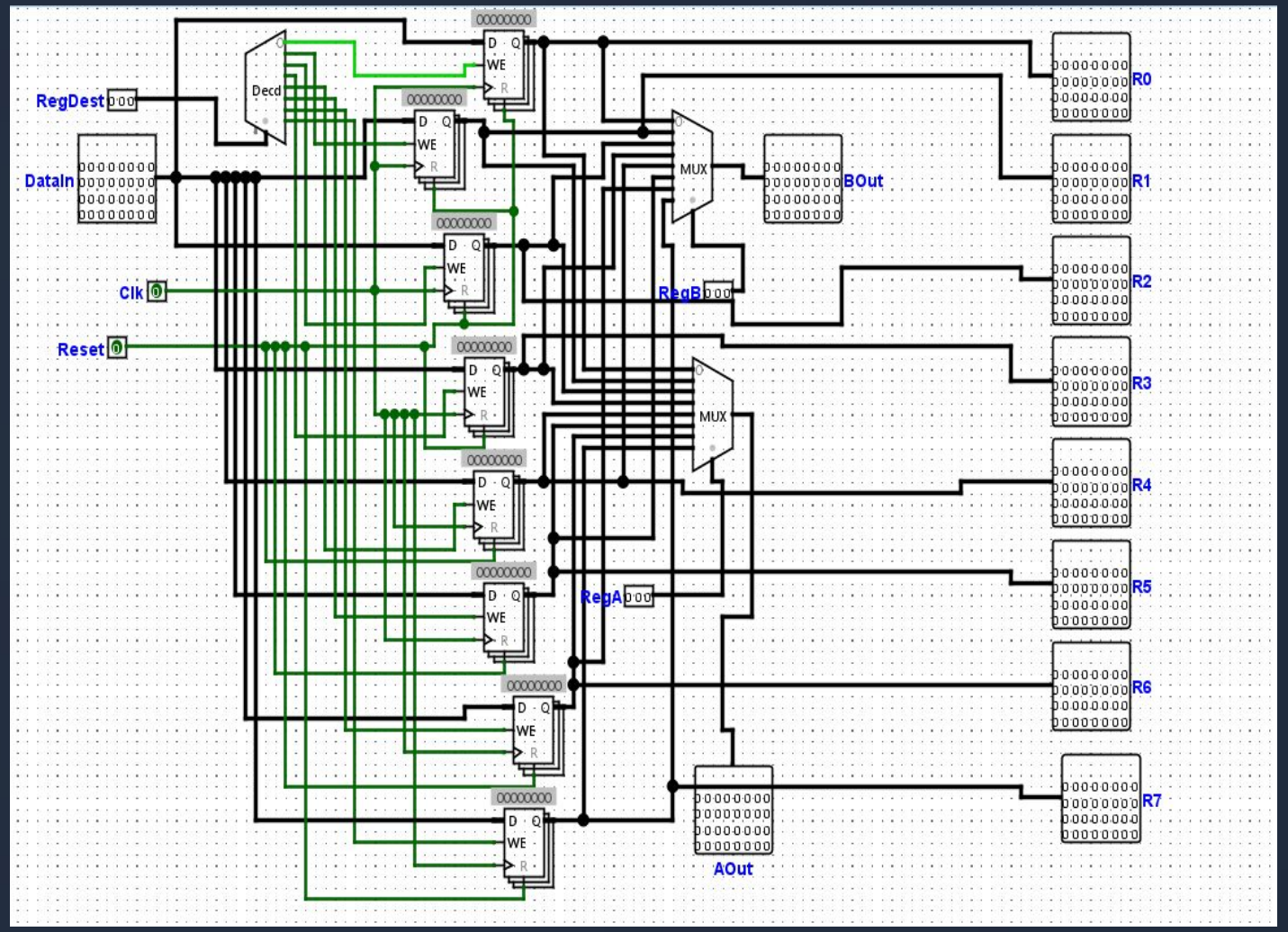
- Masquage Intelligent (FUM) : Détermination dynamique des drapeaux à modifier (NZCV, NZC, ou NZ) selon la nature de l'opération.
- Utilisation de la Masse : Neutralisation systématique des bits de drapeaux non pertinents via des splitters pour préserver l'intégrité de l'état du processeur.

Banc de registres

Stocke les registres du processeur, fournit les opérandes à l'ALU et reçoit les résultats après l'exécution des instructions.

- Stocker les registres du processeur
- Fournir les opérandes à l'ALU
- Écrire les résultats après exécution

Interface centrale entre contrôleur et ALU



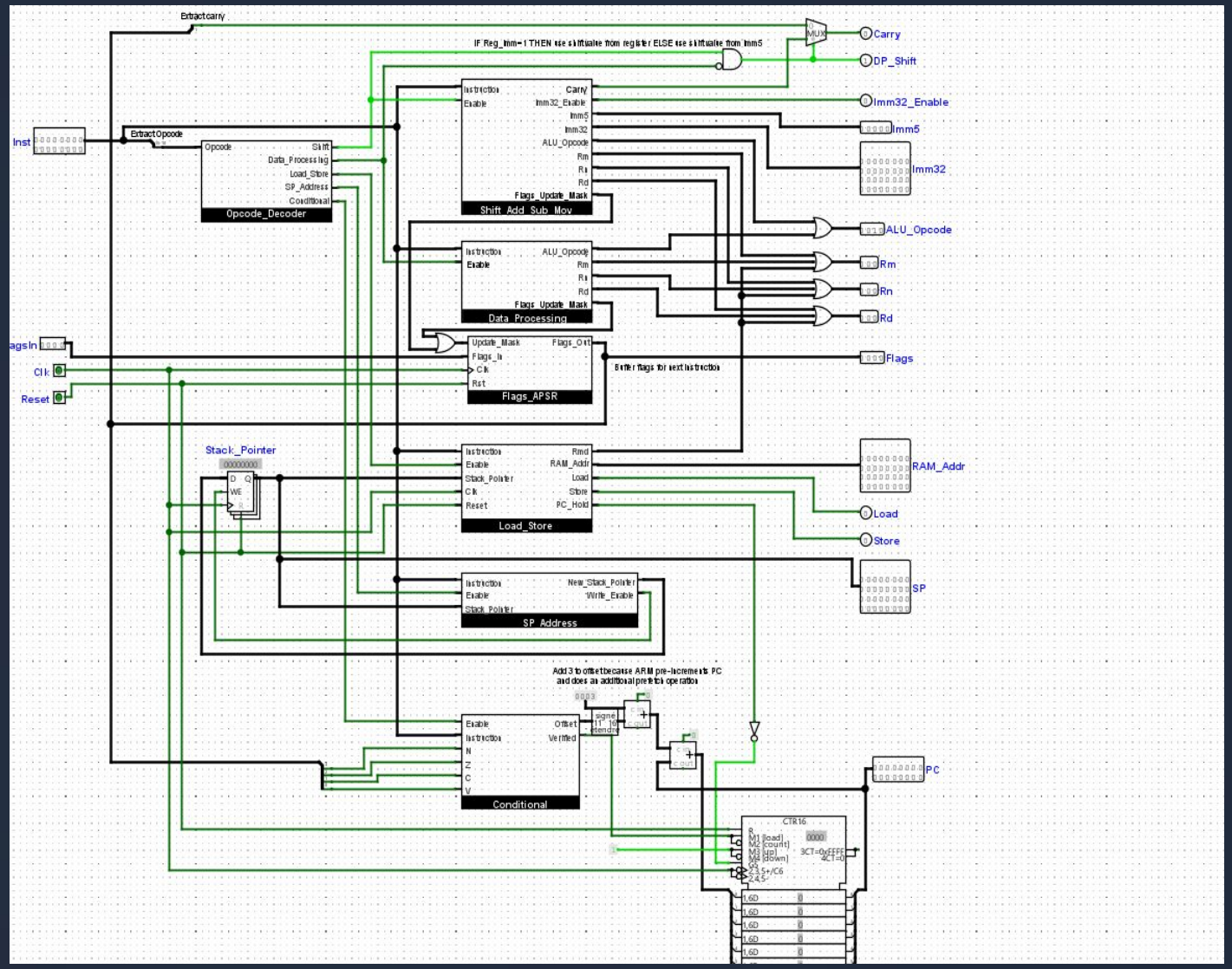
Controller

Coordination de l’exécution des instructions

Génération des signaux de contrôle

Pilotage de l’ALU, des registres et de la mémoire

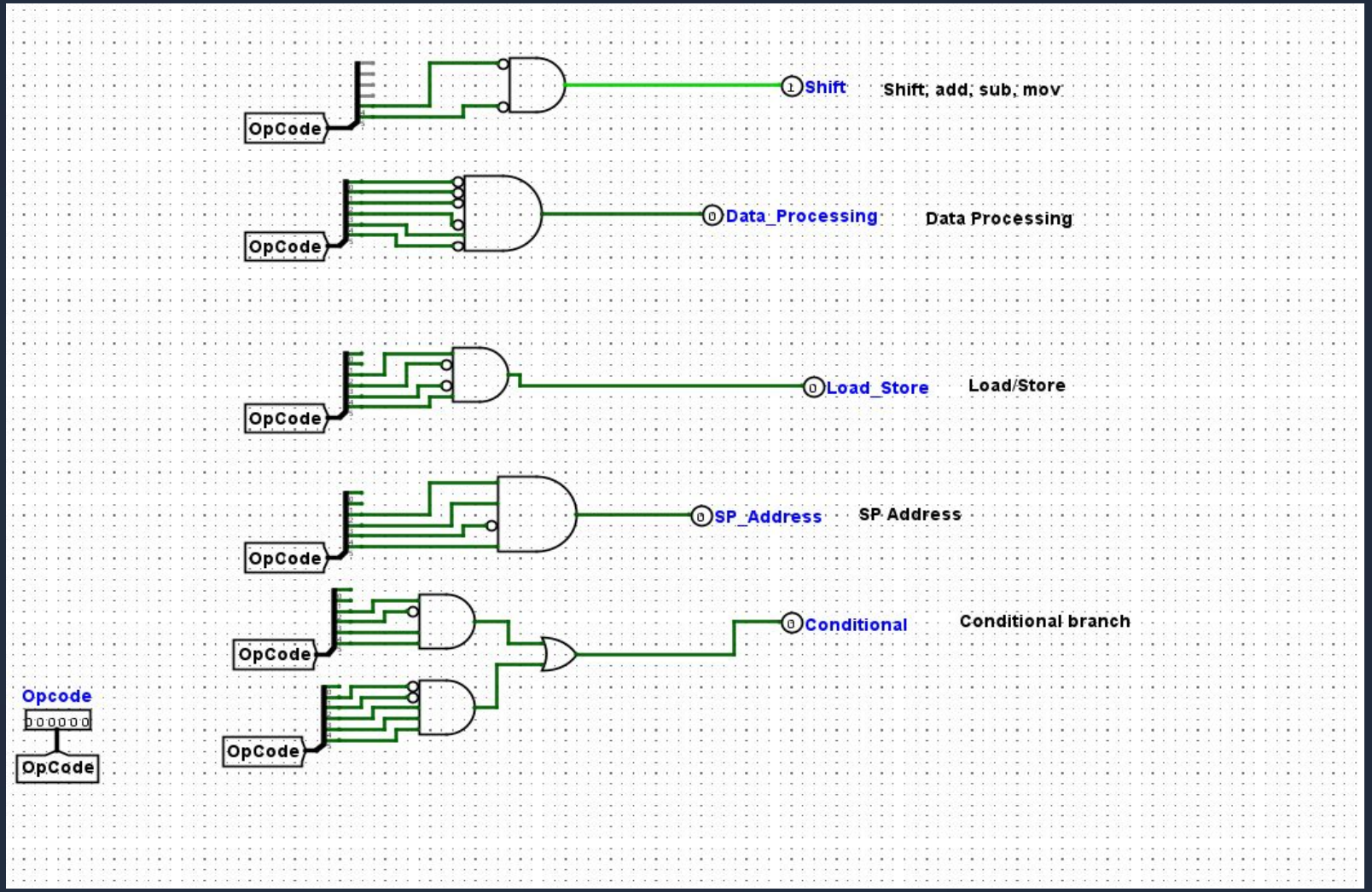
Gestion du PC et des flags



Op_Code_Decoder

Opcode Decoder

- Décodage de l’instruction
- Identification du type d’opération
- Activation des blocs de contrôle correspondants
- Génération des signaux de commande



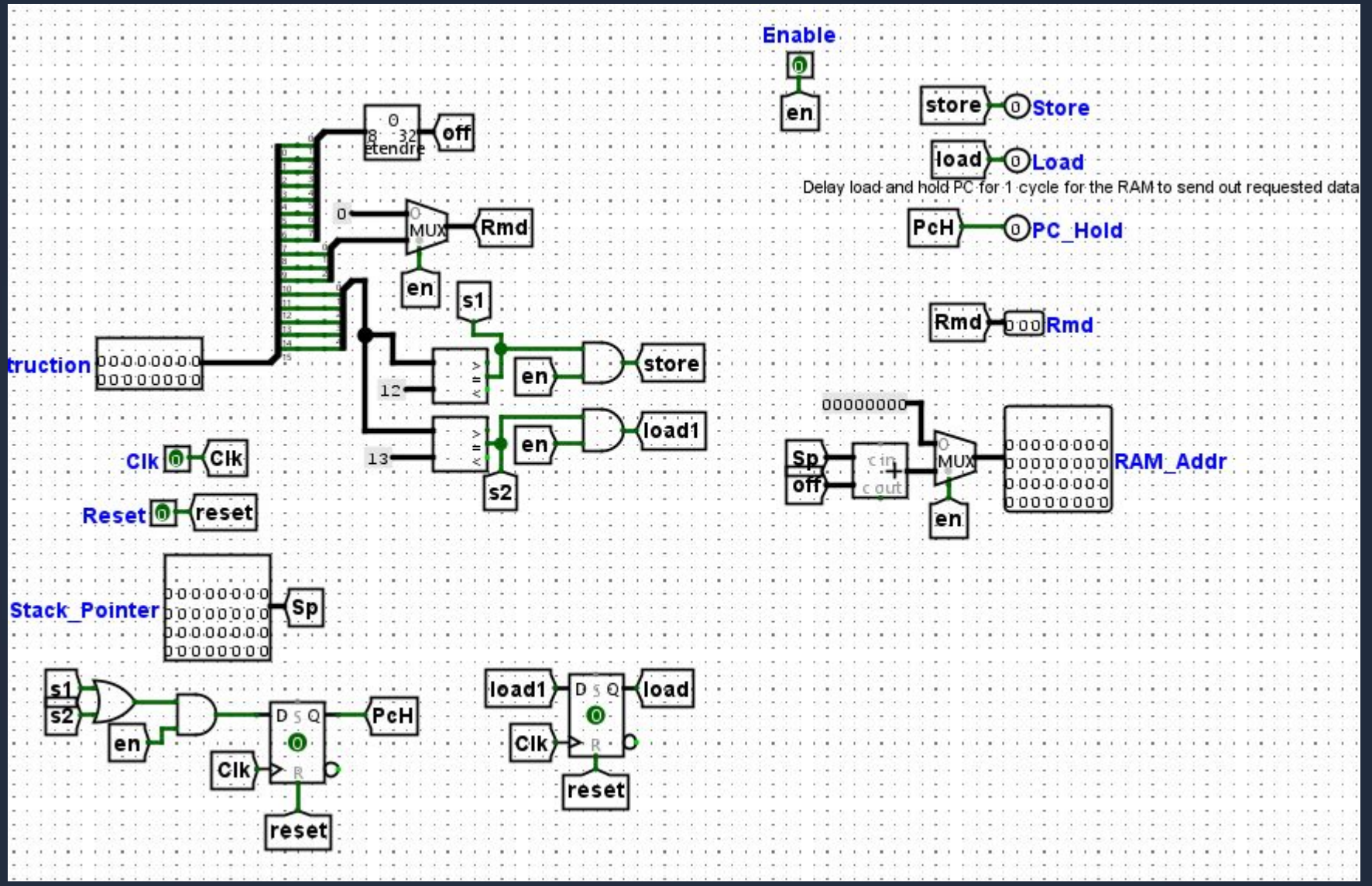
Load/Store

Accès à la mémoire

Chargement de données (LOAD)

Stockage de données (STORE)

Échange entre registres et mémoire



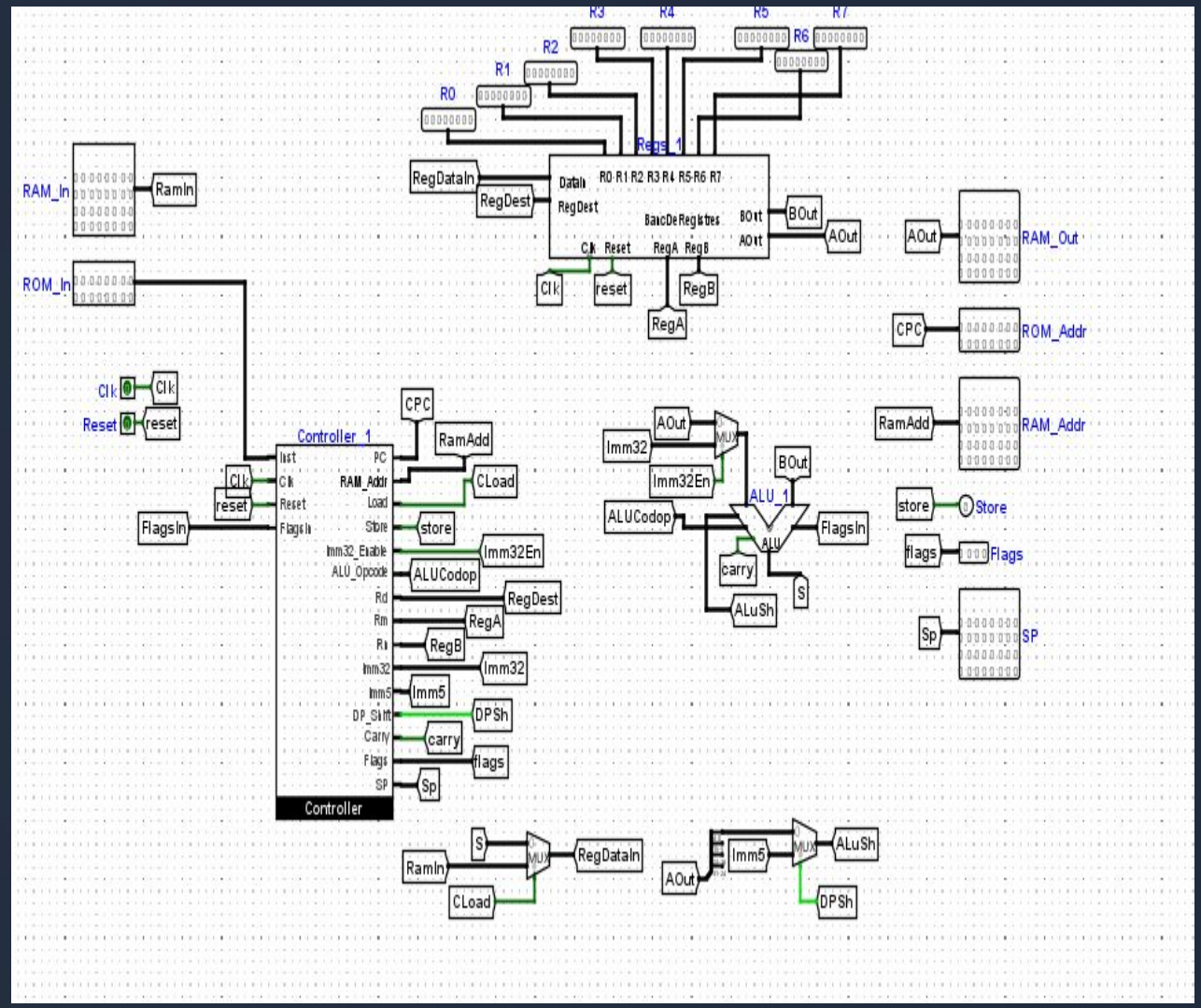
Cpu

Exécuter les instructions d’un programme

Effectuer les calculs arithmétiques et logiques

Gérer les accès aux registres et à la mémoire

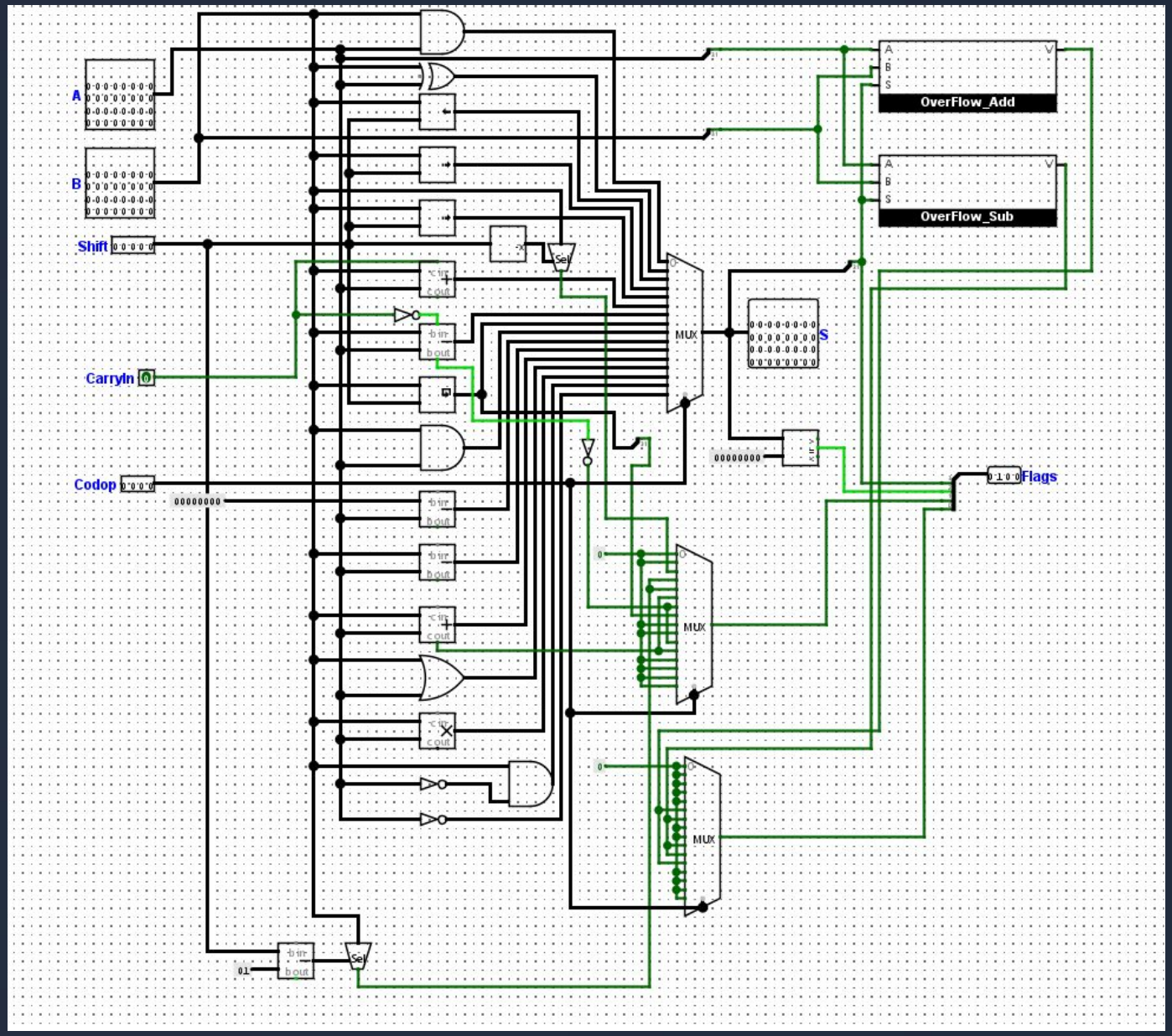
Coordonner le fonctionnement du système



ALU

L’ALU (Arithmetic Logic Unit) est le composant du processeur qui exécute les opérations arithmétiques et logiques sur les données et met à jour les indicateurs d’ état (flags).

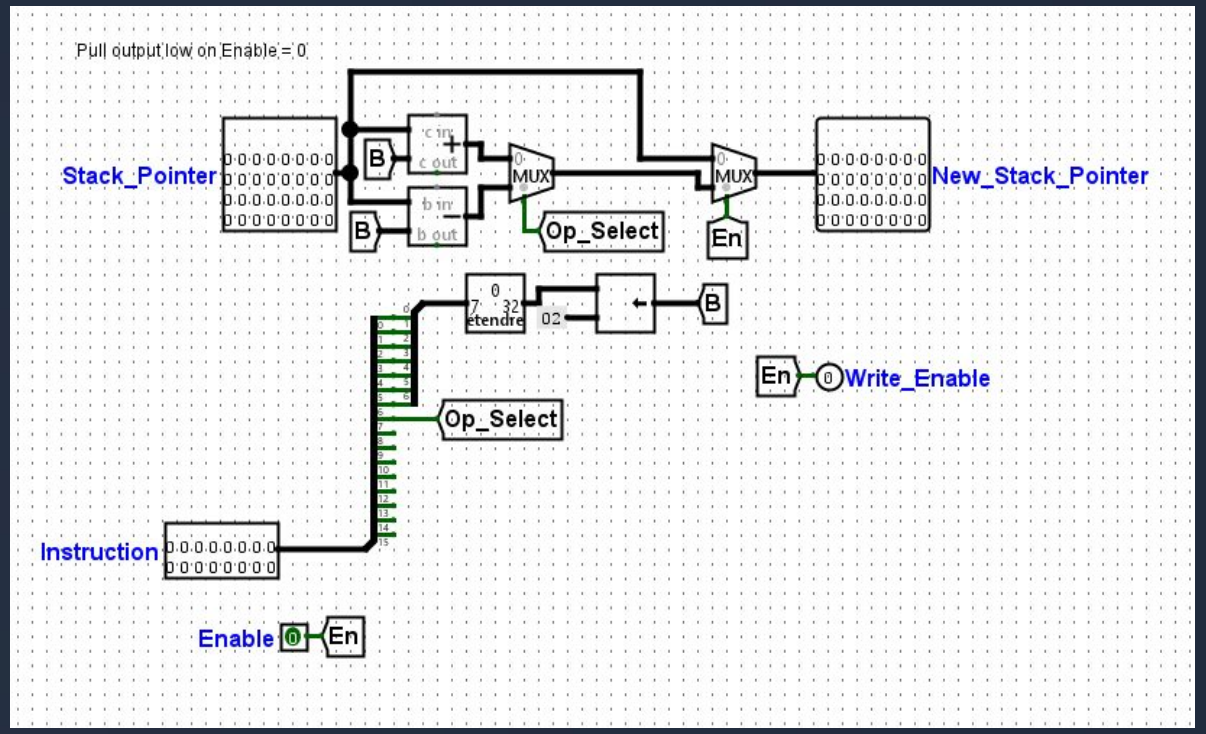
- Sélection des opérandes
- Application des décalages
- Exécution des opérations arithmétiques et logiques
- Mise à jour des flags



Gestion du Pointeur de Pile (SP Address)

- Gérer l'allocation et la désallocation dynamique de mémoire sur la pile (Stack).

- Logique de Calcul :
- Entrée : Instruction (16 bits) et SP actuel.
- Alignement : L'immédiat (7 bits) est multiplié par 4 (Décalage << 2) pour respecter l'alignement mémoire (mots de 32 bits).
- Sortie :Met à jour le registre SP uniquement si le bloc est activé.



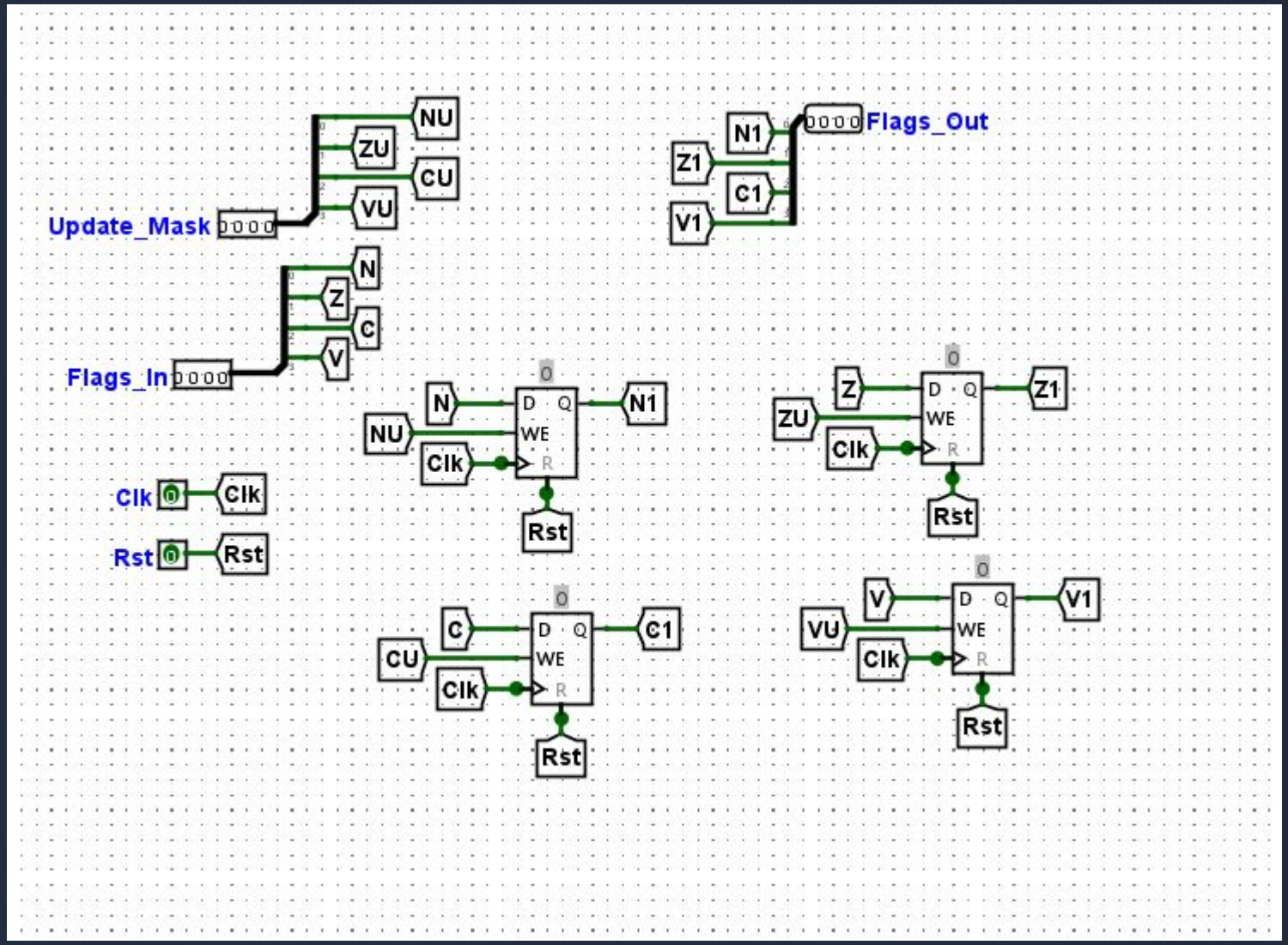
Ctl_Flags_APSR

Z : résultat nul

N : résultat négatif

C : retenue (carry)

V : dépassement arithmétique



Tests et Validation

Stratégie de Test

Assurance qualité du matériel simulé et de l'assembleur.

- **Tests Unitaires : Validation isolée de l'ALU (toutes opérations) et du décodeur.**
- **Tests d'Intégration CPU: Execution des tests sur logisim**
- **Couverture :**
  - Instructions supportées : 100%
  - Cas limites ALU : >90%
  - Parser Assembleur : 100% syntaxe valide

Alu

Réussi : 89 Erreur : 11

Statut	A	B	Shift	CarryIn	Codop	S	Flags
échec	0000 0000 0000 0000 0000 0000 0000 0000	1111 1111 1111 1111 1111 1111 1111 1111	0 0000	0	1111	1111 1111 1111 1111 1111 1111 1111 1111	1000
échec	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	1001	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	1111 1111 1111 1111 0000 0000 0000 0000	1010 1010 1010 1010 0101 0101 0101 0101	0 0000	0	0000	1010 1010 1010 1010 0000 0000 0000 0000	1000
réussit	1111 1111 1111 1111 1111 1111 1111 1111	1111 1111 1111 1111 1111 1111 1111 1111	0 0000	0	0000	1111 1111 1111 1111 1111 1111 1111 1111	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	0000	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	0000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	1100	1000 0000 0000 0000 0000 0000 0000 0000	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0010	0 0000	0	1100	0000 0000 0000 0000 0000 0000 0000 0011	0000
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	0001	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	0001	0000 0000 0000 0000 0000 0000 0000 0001	0000
réussit	1111 1111 1111 1111 1111 1111 1111 1111	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	0001	1111 1111 1111 1111 1111 1111 1111 1111	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0011	0000 0000 0000 0000 0000 0000 0000 1111	0 0000	0	1110	0000 0000 0000 0000 0000 0000 0000 1100	0000
réussit	1111 1111 1111 1111 1111 1111 1111 1111	1111 1111 1111 1111 1111 1111 1111 1111	0 0000	0	1110	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	1111	1111 1111 1111 1111 1111 1111 1111 1111	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	1000	0000 0000 0000 0000 0000 0000 0000 0001	0000
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	1000	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0001	0 0001	0	0010	0000 0000 0000 0000 0000 0000 0000 0010	0000
réussit	0000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000	0 0001	0	0010	0000 0000 0000 0000 0000 0000 0000 0000	0110
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0010	0 0001	0	0011	0000 0000 0000 0000 0000 0000 0000 0001	0000
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0001	0 0001	0	0011	0000 0000 0000 0000 0000 0000 0000 0000	0110
réussit	0000 0000 0000 0000 0000 0000 0000 0000	1111 1111 1111 1111 1111 1111 1111 1110	0 0001	0	0100	1111 1111 1111 1111 1111 1111 1111 1111	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0000	1111 1111 1111 1111 1111 1111 1111 1111	0 0001	0	0100	1111 1111 1111 1111 1111 1111 1111 1111	1010
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0001	0 0001	0	0111	1000 0000 0000 0000 0000 0000 0000 0000	1010
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0011	0 0001	0	0111	1000 0000 0000 0000 0000 0000 0000 0001	1010
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0010	0 0000	0	0101	0000 0000 0000 0000 0000 0000 0000 0011	0000
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	0101	0000 0000 0000 0000 0000 0000 0000 0000	0100
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0010	0 0000	1	0101	0000 0000 0000 0000 0000 0000 0000 0100	0000
réussit	1111 1111 1111 1111 1111 1111 1111 1111	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	0101	0000 0000 0000 0000 0000 0000 0000 0000	0110
réussit	0111 1111 1111 1111 1111 1111 1111 1111	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	0101	1000 0000 0000 0000 0000 0000 0000 0000	1001
réussit	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0010	0 0000	0	1011	0000 0000 0000 0000 0000 0000 0000 0011	0000
réussit	1111 1111 1111 1111 1111 1111 1111 1111	0000 0000 0000 0000 0000 0000 0000 0001	0 0000	0	1011	0000 0000 0000 0000 0000 0000 0000 0000	0110
réussit	0000 0000 0000 0000 0000 0000 0000 0101	0000 0000 0000 0000 0000 0000 0000 1010	0 0000	0	1010	0000 0000 0000 0000 0000 0000 0000 0101	0010
réussit	0000 0000 0000 0000 0000 0000 0000 1010	0000 0000 0000 0000 0000 0000 0000 0101	0 0000	0	1010	1111 1111 1111 1111 1111 1111 1111 1011	1000
réussit	0000 0000 0000 0000 0000 0000 0000 0001	1000 0000 0000 0000 0000 0000 0000 0000	0 0000	0	1010	0111 1111 1111 1111 1111 1111 1111 1111	0011
réussit	0000 0000 0000 0000 0000 0000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0101	0 0000	1	0110	0000 0000 0000 0000 0000 0000 0000 0100	0010

Tests et Validation

Data_Processing

re Aide Logisim : Vecteur de test Data_Processing de Ctl_Data_Processing

Réussi : 8 Erreur : 0

Statut	Instruction	Enable	ALU_Opcode	Rm	Rn	Rd	Flags_Update_Mask
réussit	0000 0000 1011 0101	1	0010	110	101	101	1110
réussit	0000 0000 1100 1110	1	0011	001	110	110	1110
réussit	0000 0000 1101 0001	1	0011	010	001	001	1110
réussit	0000 0000 1000 0000	1	0010	000	000	000	1110
réussit	0000 0000 1001 0001	1	0010	010	001	001	1110
réussit	0000 0000 1110 0110	1	0011	100	110	110	1110
réussit	0000 0000 1111 1001	1	0011	111	001	001	1110
réussit	0000 0000 1010 0011	0	0000	000	000	000	0000

Ctl_SP_Address

Réussi : 5 Erreur : 0

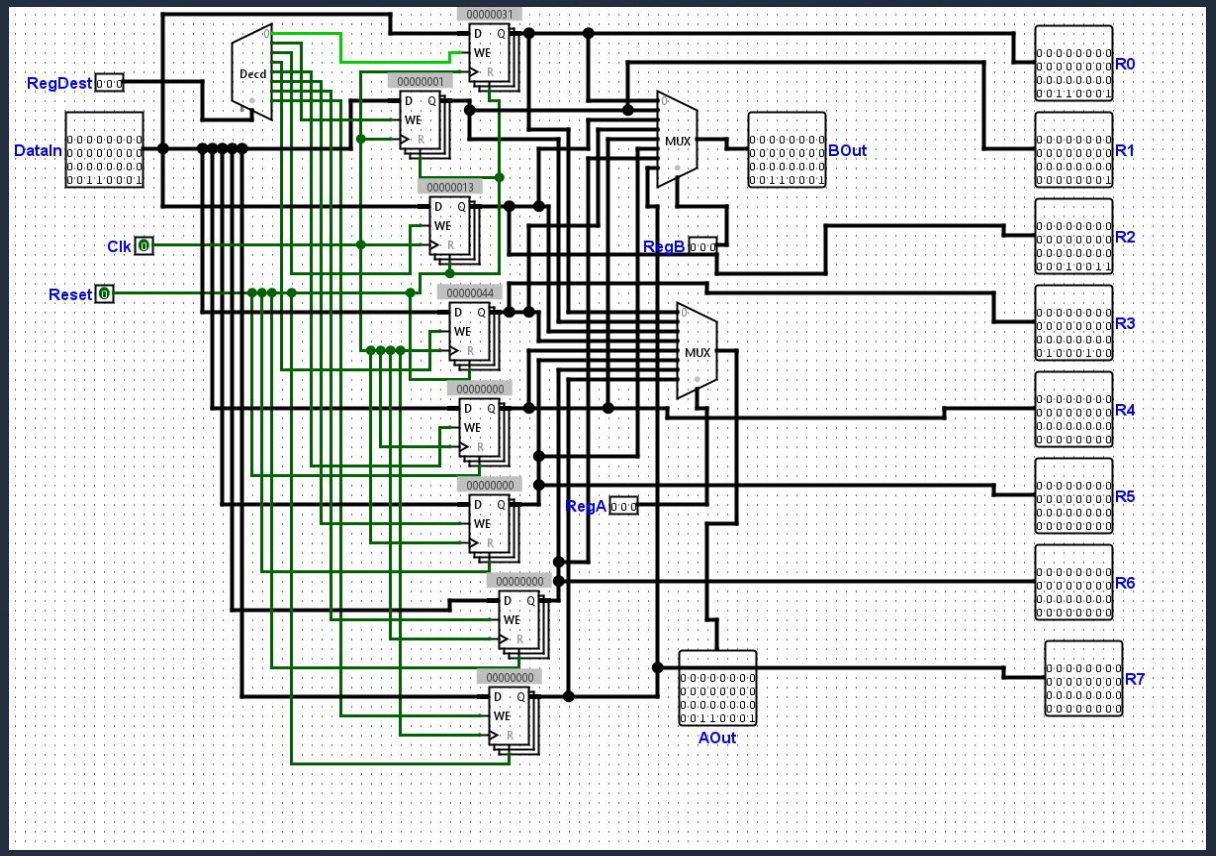
Statut	Instruction	Enable	Stack_Pointer	Write_Enable	New_Stack_Pointer
réussit	1011 0000 0000 0001	1	0000 0000 0000 0000 0000 0000 0001 0000	1	0000 0000 0000 0000 0000 0000 0001 0100
réussit	1011 0000 0000 0100	1	0000 0000 0000 0000 0000 0000 0001 0000	1	0000 0000 0000 0000 0000 0000 0010 0000
réussit	1011 0000 1000 0001	1	0000 0000 0000 0000 0000 0000 0001 0000	1	0000 0000 0000 0000 0000 0000 0000 1100
réussit	1011 0000 1000 0101	1	0000 0000 0000 0000 0000 0000 0001 0100	1	0000 0000 0000 0000 0000 0000 0000 0000
réussit	1011 0000 0000 0001	0	0000 0000 0000 0000 0000 0000 0001 0000	0	0000 0000 0000 0000 0000 0000 0001 0000

Tests et Validation

Shift_Add_Sub_Mov

Statut	Instruction	Enable	ALU_Opcode	Rn	Rm	Rd	Imm32_Enable	Carry	Flags_Update_Mask
échec	0000 0001 0001 0010	1	0010	010	uuu	010	0	0	1110
échec	0010 0100 1010 1010	1	1001	100	uuu	100	1	0	1100
échec	0010 1100 0000 1111	1	1010	100	uuu	100	1	0	1111
réussit	0000 1000 0100 1010	1	0011	001	uuu	010	0	0	1110
réussit	0001 0001 0010 0011	1	0100	100	uuu	011	0	0	1110
réussit	0001 1001 1111 0101	1	0101	110	111	101	0	0	1111
réussit	0001 1010 1000 1000	1	0110	001	010	000	0	1	1111
réussit	0001 1101 0101 1010	1	0101	011	uuu	010	1	0	1111
réussit	0001 1111 1100 0001	1	0110	000	uuu	001	1	1	1111
réussit	0011 0010 0001 0000	1	0101	010	uuu	010	1	0	1111
réussit	0011 1010 0000 0001	1	0110	010	uuu	010	1	1	1111
réussit	0010 0100 1010 1010	0	0000	000	000	000	0	0	0000

Test integration

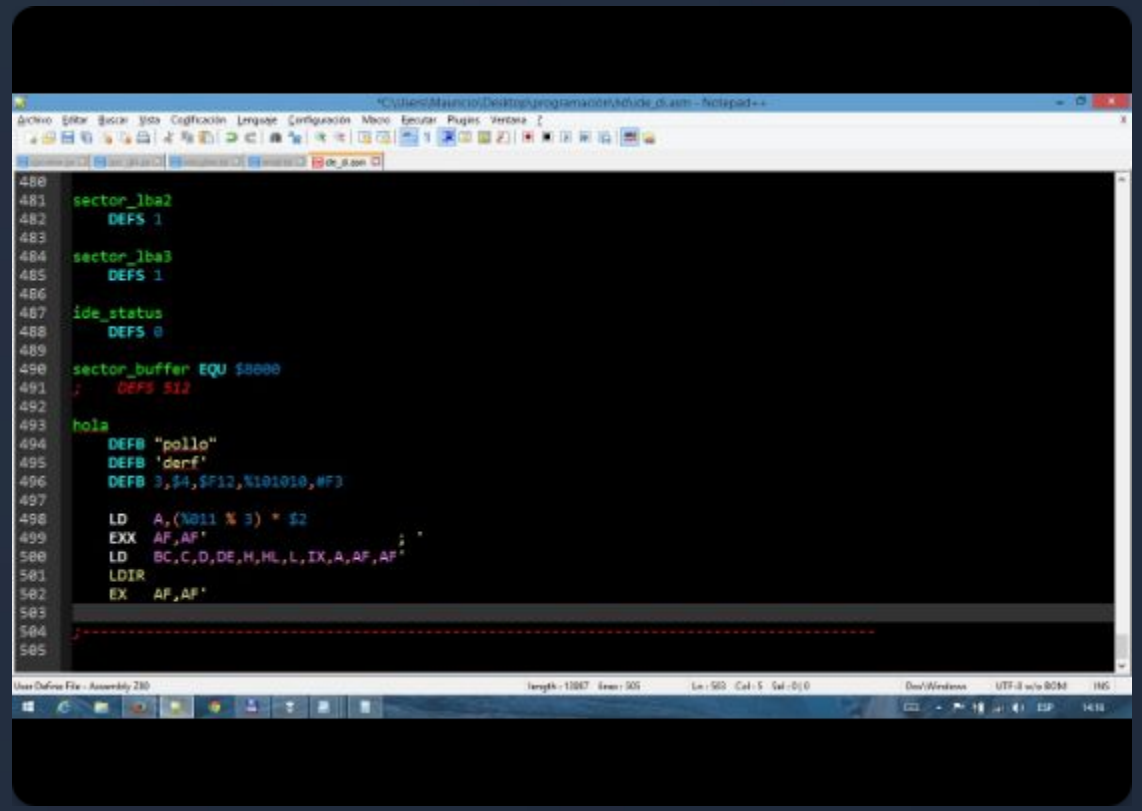


Chaîne Logicielle

De l'Assembleur au Binaire

Développement d'outils pour traduire le code humain en langage machine Logisim.

- **Parser :** Analyse syntaxique des mnémoniques (MOV, ADD, LDR).
- **Labels :** Gestion des étiquettes pour les sauts et boucles.
- **Output :** Génération de fichiers hexadécimaux/binaires chargeables directement dans la RAM/ROM Logisim



Tests d’intégration et compilations C (Assembleur)

Stratégie de Test

Assurance qualité du matériel simulé et de l'assembleur.

- **Tests d'Intégration parseur :** Exécution de programmes complets (Fibonacci, Factorielle).
- **Tests de compilation C :** Compilation des programmes C

- **Couverture :**
  - Instructions supportées : 89%
  - Parser Assembleur : 97% syntaxe valide (estimation)

Conclusion et Perspectives