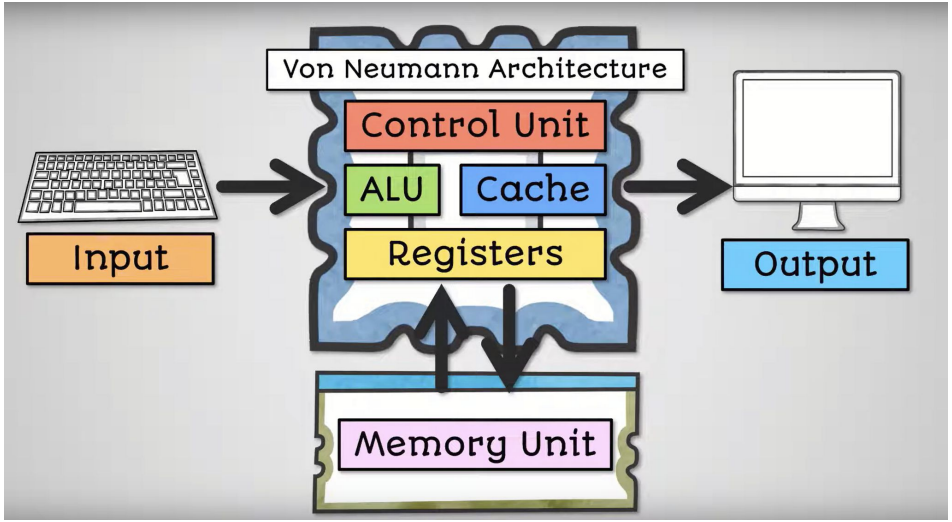
PARM354N

ARM based processor

Patrick Bizot, Deyann Koperecz, Lohann Colle, Zaynab Nachabe

ARM: Cortex-M0 Processor simulation using Logisim

Le PARM354N est un processeur informatique à jeu d'instructions réduit de environ 50 instruction avec une architecture Von-Neumann. Les instructions sont générées par le compilateur C, en utilisant l'instruction 16 bits. Le processeur PARM354N est très performant car le jeu d'instructions est très optimisé. Il est classé comme architecture load-store, car il possède des instructions séparées pour la lecture et l'écriture dans la mémoire, et des instructions pour les opérations arithmétiques ou logiques qui utilisent des registres.

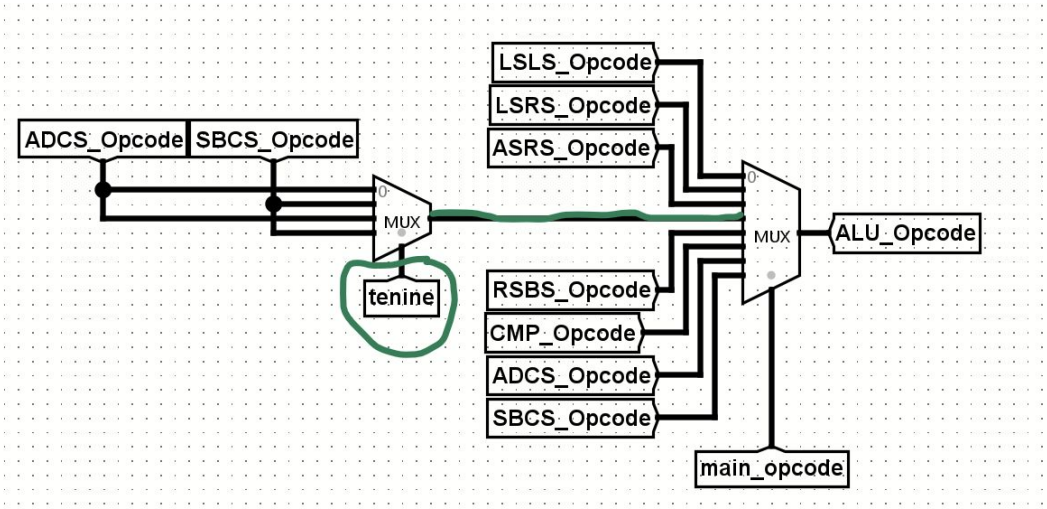


Quelques uns de nos circuits

Shift, Add, Sub, Move

ALU Opcode

Pour déterminer le bon ALU_Opcode pour chaque instruction, nous découpons l'instruction en plusieurs parties distinctes, chacune correspondant à une opération spécifique. Dans le cas des instructions ADD et SUB, leurs codes d'opération ont une structure commune, mais diffèrent aux bits 10 et 9. Le main_opcode est défini par les bits 11 à 13.



Shift, Add, Sub, Move

0010	B << Shift	LSL	Retenue sortante, voir jeu d'instructions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Rm			Rd		

0011	B >> Shift	LSR	Retenue sortante, voir jeu d'instructions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	imm5					Rm			Rd		

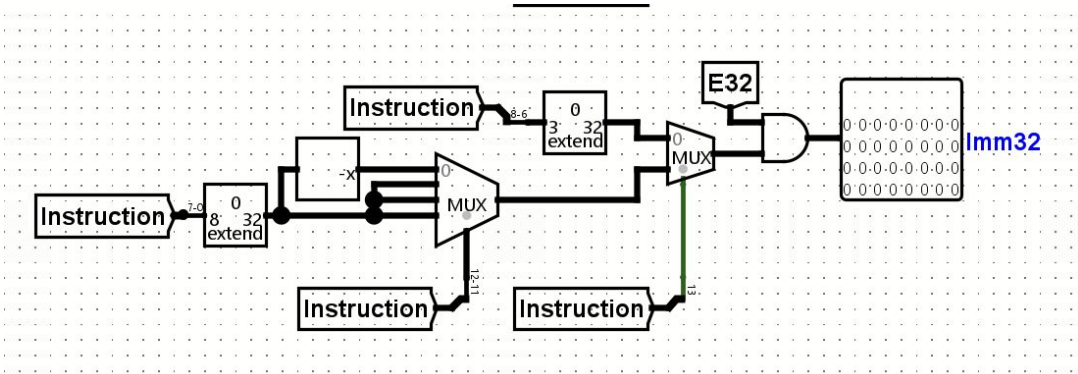
0100	B >> Shift (arith)	ASR	Retenue sortante, voir jeu d'instructions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	imm5					Rm			Rd		

Shift, Add, Sub, Move

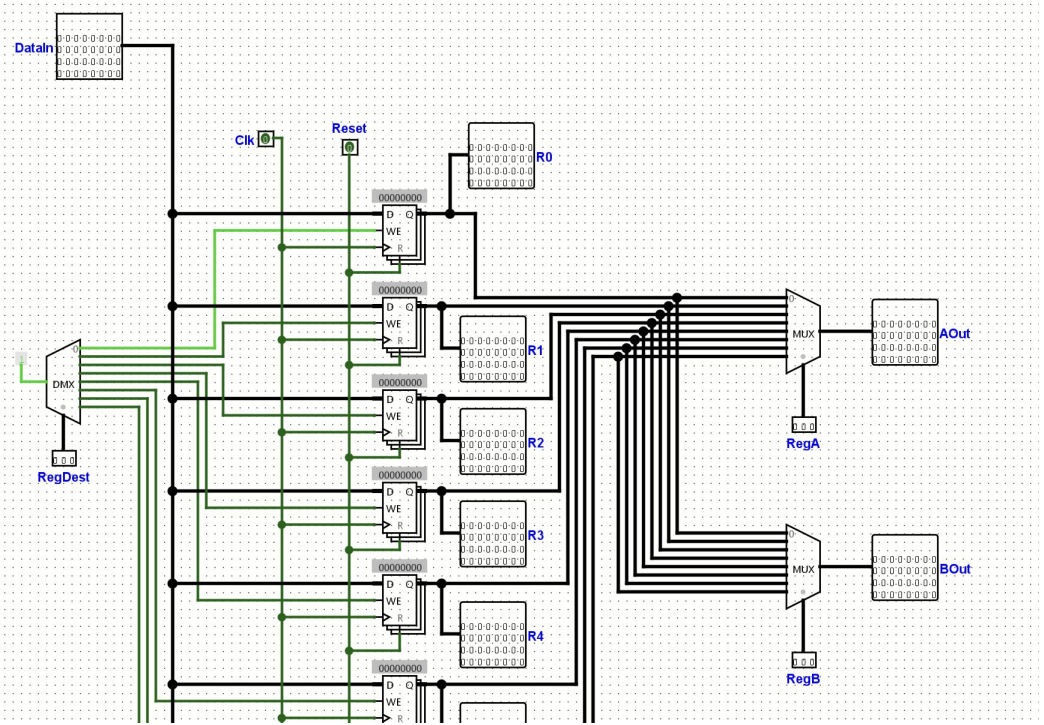
Imm32

Les bits extraits de **Instruction** sont combinés pour former la valeur immédiate de 32 bits. On étend les bit 0-7 et les bits 8-6 en 32 bits. On fait une négation de l'Imm32 à fin de pouvoir utiliser l'instruction RSB de l'ALU.



Banc de registres

- Stocke les données dans des registres auxquels le processeur peut accéder rapidement.
- Récupère les données des registres pour les utiliser dans diverses opérations.
- Achemine les signaux de commande pour sélectionner le registre approprié pour la lecture ou l'écriture.
- Assure que les opérations de stockage et d'extraction des données sont synchronisées avec l'horloge du système.
- Utilise des multiplexeurs pour sélectionner le registre approprié en fonction des signaux de commande.



Flag APSR - La Doc

6.3.4 Flags APSR

6.3.4.1 Description

Le bloc *Flags APSR* correspond au 4 bits de poids fort du registre `Application Program Status Register` de l'architecture ARM. Il conserve les drapeaux générés par la dernière instruction, afin qu'ils soient disponibles pour la prochaine instruction.

L'entrée `Update_Mask` permet de réinjecter l'ancien état d'un drapeau si le bit correspondant est à 0. Si le bit est à 1, le drapeau est mis à jour.

6.3.4.2 Interface

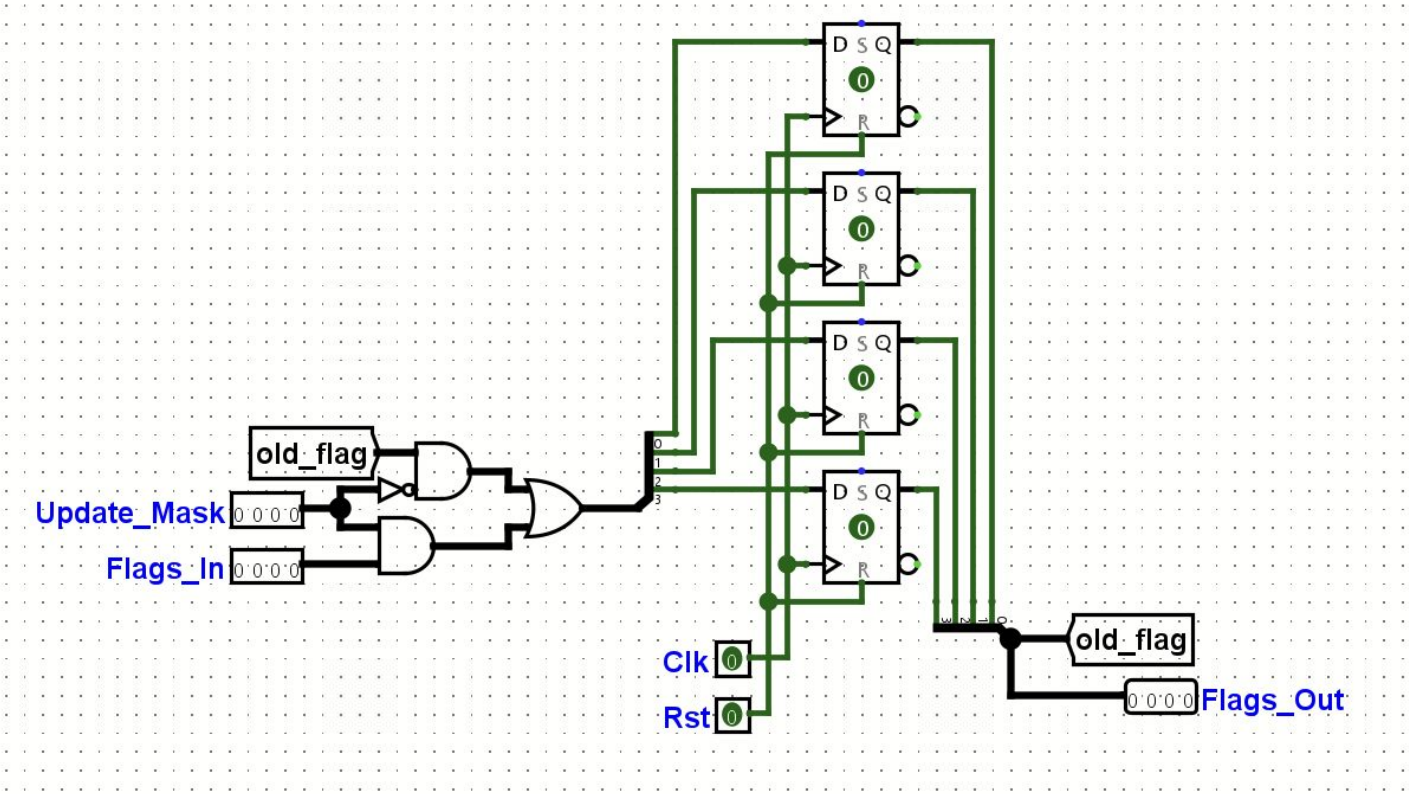
Entrées

Port	Taille	Description
`Flags_In`	4	Nouveaux drapeaux générés par l'instruction courante, ordre `NZCV`
`Update_Mask`	4	Masque d'enregistrement des drapeaux, ordre `NZCV`
`Clk`	1	Horloge
`Reset`	1	Remise à zéro

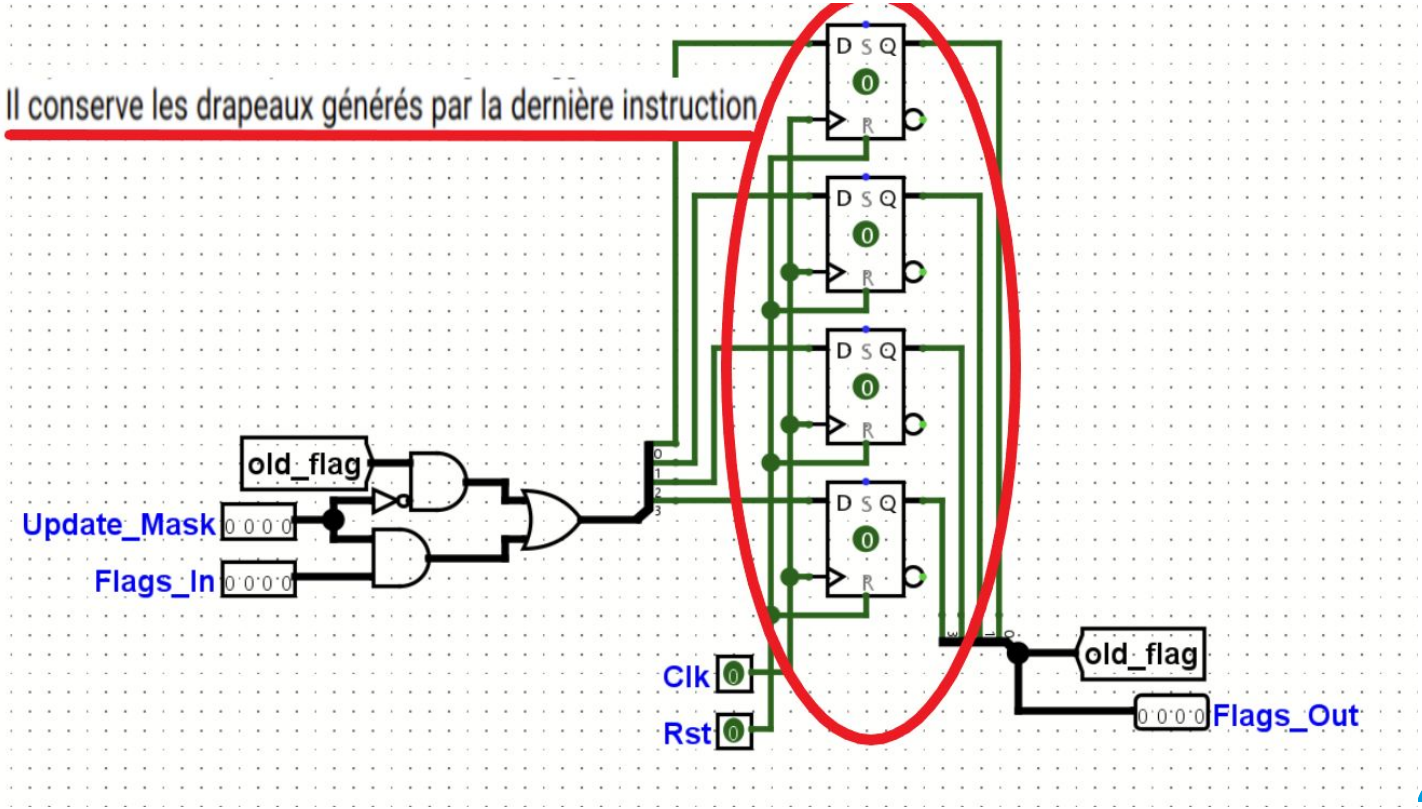
Sorties

Port	Taille	Description
`Flags_Out`	4	Drapeaux générés par l'instruction précédente, ordre `NZCV`

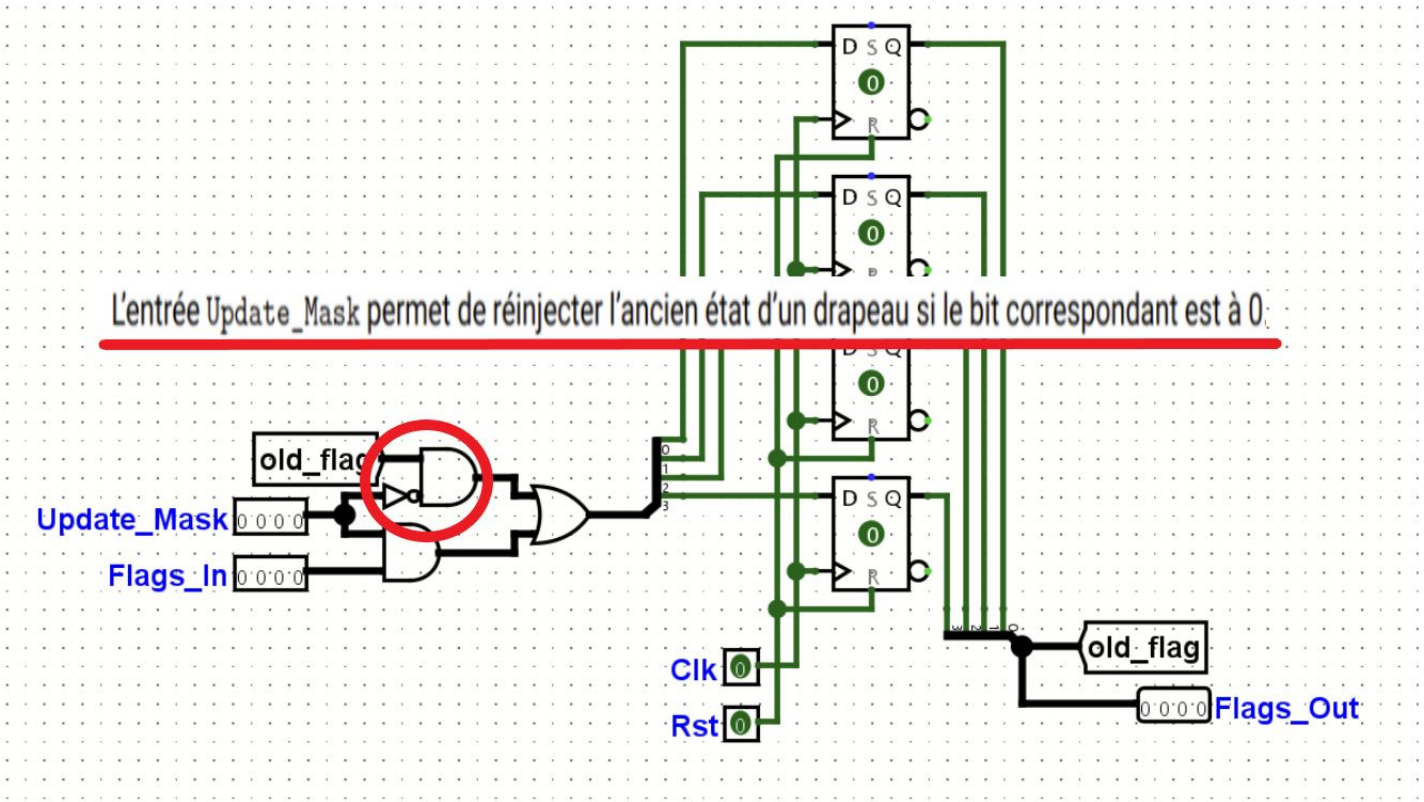
Flag APSR


D S Q
0
R
old_flag
Update_Mask
0 0 0 0
Flags_In
0 0 0 0
Clk
0
Rst
0
old_flag
0 0 0 0
Flags_Out

Flag APSR


Il conserve les drapeaux générés par la dernière instruction
D S Q
R
old_flag
Update_Mask
Flags_In
Clk
Rst
old_flag
Flags_Out

Flag APSR


L'entrée Update_Mask permet de réinjecter l'ancien état d'un drapeau si le bit correspondant est à 0
old_flag
Update_Mask
Flags_In
Clk
Rst
old_flag
Flags_Out

Flag APSR

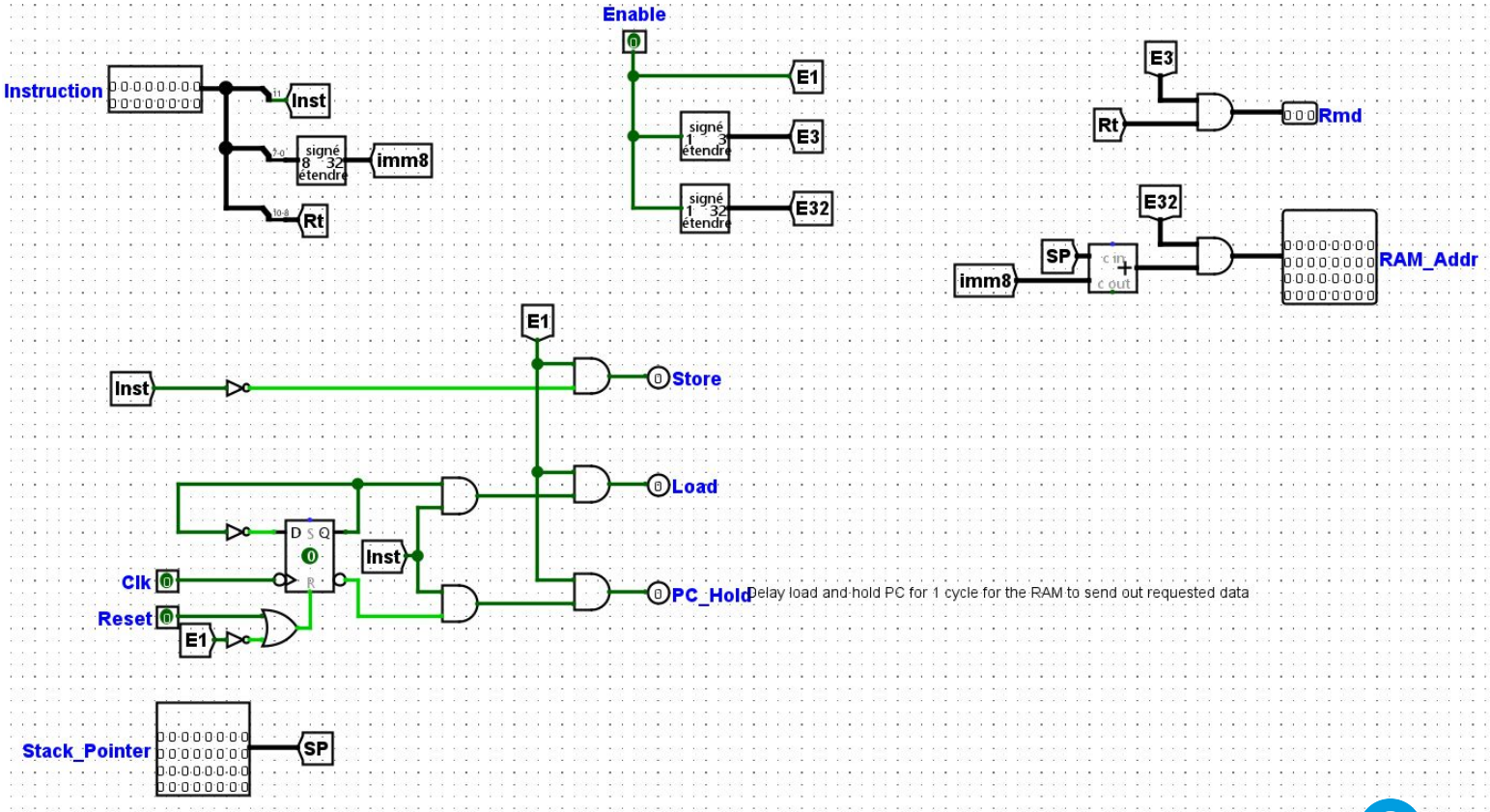
Bascules D :

+Plus petites
+Plus jolies
+Il y a un autre tunnel

Registres :

-Gros
-Gros
-Gros
-Gros
-Gros

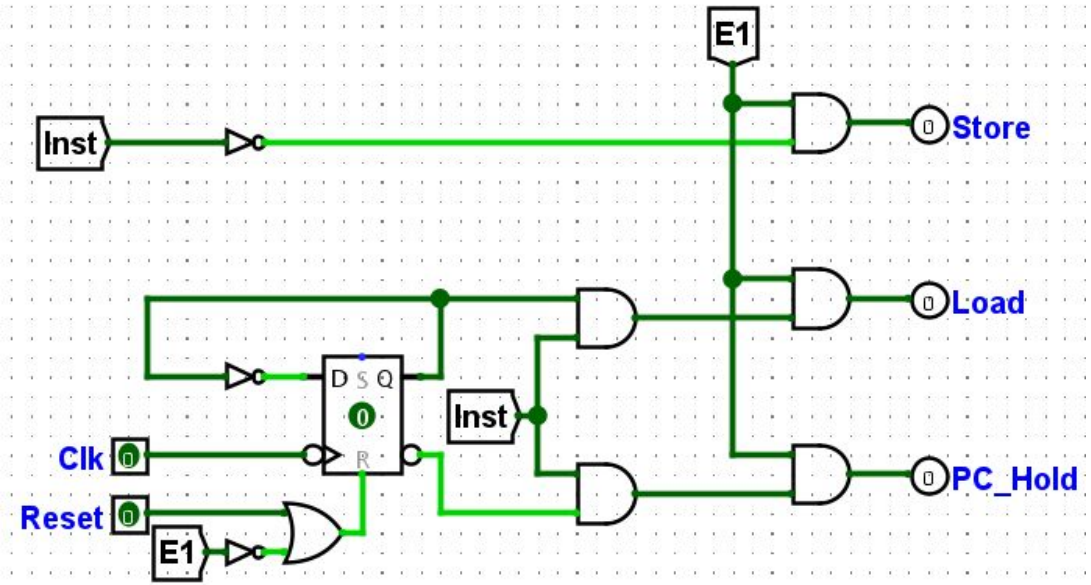
Load/Store


Instruction
Inst
signé 8 32 étendre
imm8
Rt
Enable
E1
signé 1 3 étendre
E3
signé 1 32 étendre
E32
E3
Rt
Rmd
E32
SP
imm8
c in
c out
RAM_Addr
E1
Inst
Store
Load
D S Q
R
Inst
Clk
Reset
E1
PC_Hold
Delay load and hold PC for 1 cycle for the RAM to send out requested data
Stack_Pointer
SP

Load/Store

- Fait le liens entre la RAM et le CPU
- Consiste en 2 instructions :
  - **Store** : Copie la valeur d'un registre dans la RAM
  - **Load** : Copie une valeur de la RAM dans un registre
- Assure la cohérence entre la mémoire et l'exécution du programme en retardant l'exécution le temps que les données aient été acheminées de la RAM au CPU (lors d'un Load).

Load/Store



1er coup de clock	2ème coup de clock
E1 : 1	E1 : 1
Bascule (av) : 0	Bascule (av) : 1
Store : 0 **Load :** 0 **PC_Hold :** 1	**Store :** 0 **Load :** 1 **PC_Hold :** 0
Bascule (ap) : 1	Bascule (ap) : 0

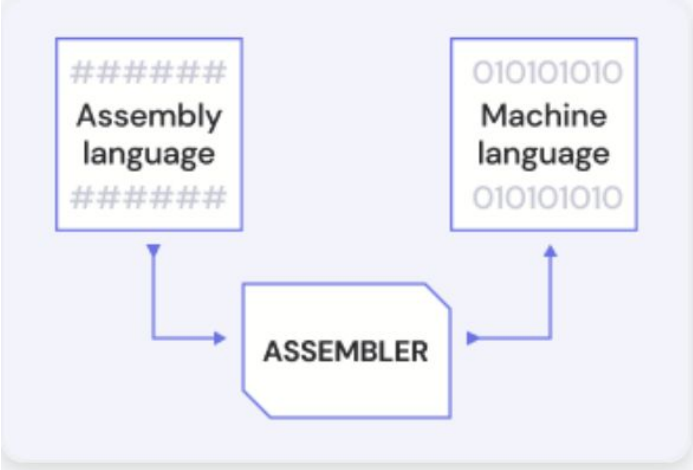
L’assembleur

Assembleur

Parseur de texte:
Transforme les commandes écrites en langage Assembleur en leur équivalent en langage Machine.

Deux étapes:

- Une passe pour les conditionnels
- Une passe pour la transformation en Hexadécimal



Les Tests

Tests unitaires

Tests unitaires pour tous les les composants séquentiels :

- ALU
- CTL_Conditional
- CTL_Data_processing
- CTL_Shift_Add_Sub_Mov
- CTL_SP address.

Mais surtout, des tests unitaires très robustes sur l'ALU nous permettant de cibler exactement qu'elle opérations arithmétique ou logique a des problèmes.

Tous nos tests unitaires passent, ce qui nous donne une certaine confiance.

Tests d’integration

Tous les tests d’intégration passent !!!

Tests de l’assembleur

Consiste en un assemblage des codes en langage assembleurs fournis dans les dossiers **code_c** et **code_asm**.

Puis une vérification de la cohérences des résultats obtenus en exécutant ces codes assemblés sur Logisim et en les comparant avec les codes binaires (en héxa) attendus.

Organisation

Patrick : Conditional, Load/Store, Data processing, SASM, ALU, Tests d’intégration.

Lohann : il existe et il a fait l’assembleur.

Deyann : stack pointer, Opcode spreadsheet, SP address, Flags, ALU, Tests d’intégration.

Zaynab : Banc de registres, Décodeur d’instruction, partie du SASM (ALU_Opcode decoder), Slides, Test d’intégration.

Merci pour votre attention