Projet P-ARM

Equipe : Parm Project

Sagesse ADABADJI
Selom Ami ADZAHO
Robin PERLES
Sara TAOUFIQ

Sommaire

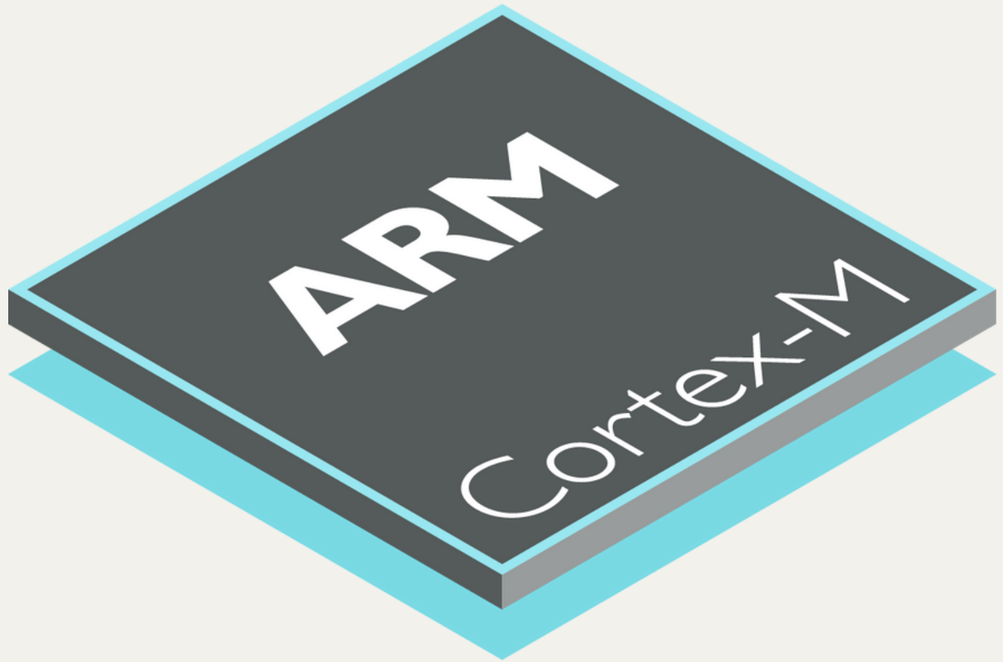
Introduction

Simulation Simplifiée

Partie materielle

Partie logiciel

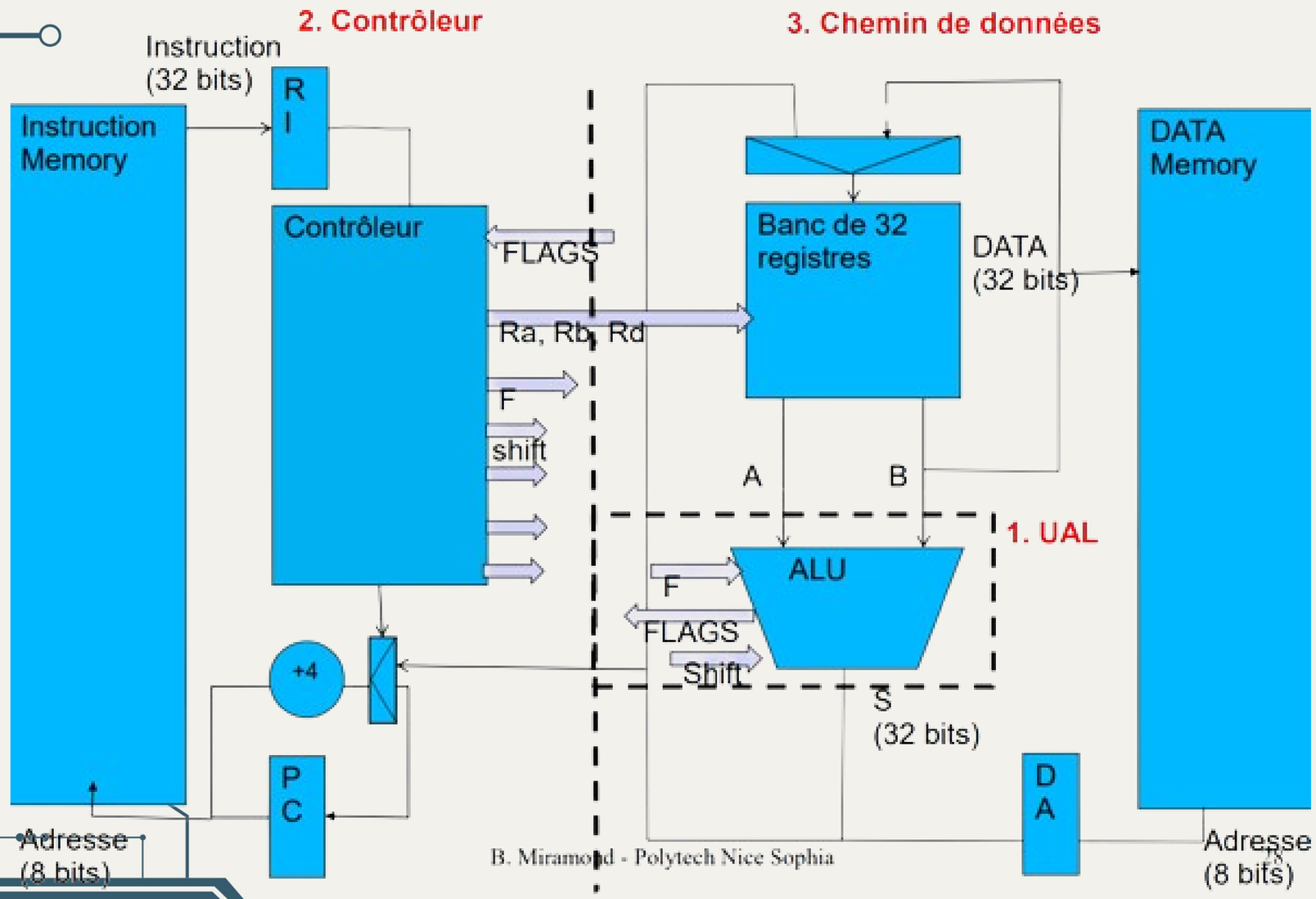
Logisim



Cortex-M0

pngwing.com

Contexte


2. Contrôleur
3. Chemin de données
Instruction
(32 bits)
R
I
Instruction
Memory
Contrôleur
FLAGS
Ra, Rb, Rd
F
shift
Banc de 32
registres
DATA
(32 bits)
DATA
Memory
A
B
1. UAL
ALU
F
FLAGS
Shift
S
(32 bits)
+4
P
C
D
A
Adresse
(8 bits)
Adresse
(8 bits)
B. Miramond - Polytech Nice Sophia

Description	UAL code				Bits															Flags			
	Instruction	Operandes			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	C	N	Z	V
Logical Shift Left	LSLS	<Rd>,	<Rm>	#<imm5>	0	0	0	0	0	imm5					Rm			Rd			x		x	x
Logical Shift Right	LSRS	<Rd>,	<Rm>	#<imm5>	0	0	0	0	1	imm5					Rm			Rd					x	x
Arithmetic Shift Right	ASRS	<Rd>,	<Rm>	#<imm5>	0	0	0	1	0	imm5					Rm			Rd			x		x	x
Shift, add , sub , mov					0	0	OpCode																	
Add register	ADDS	<Rd>,	<Rn>	<Rm>	0	0	0	1	1	0	0	Rm			Rn			Rd			x	x	x	x
Substract Register	SUBS	<Rd>,	<Rn>	<Rm>	0	0	0	1	1	0	1	Rm			Rn			Rd			x	x	x	x
Add 3-bit immediate	ADDS	<Rd>,	<Rn>	<imm3>	0	0	0	1	1	1	0	imm3			Rn			Rd			x	x	x	x
Substract 3-bit immediate	SUBS	<Rd>,	<Rn>	<imm3>	0	0	0	1	1	1	1	imm3			Rn			Rd			x	x	x	x
Move	MOVS	<Rd>,	#<imm8>		0	0	1	0	0	Rd		imm8											x	x
Data Processing					0	1	0	0	0	0	OpCode													
Bitwise AND	ANDS	<RDn>,	<Rm>		0	1	0	0	0	0	0	0	0	0	Rm			Rdn					x	x
Exlusive OR	EORS	<RDn>,	<Rm>		0	1	0	0	0	0	0	0	0	1	Rm			Rdn					x	x
Logical Shift Left	LSLS	<RDn>,	<Rm>		0	1	0	0	0	0	0	0	1	0	Rm			Rdn			x		x	x
Logical shift right	LSRS	<RDn>,	<Rm>		0	1	0	0	0	0	0	0	1	1	Rm			Rdn					x	x
Arithmetic Shift right	ASRS	<RDn>,	<Rm>		0	1	0	0	0	0	0	1	0	0	Rm			Rdn			x		x	x
Add With Carry	ADCS	<RDn>,	<Rm>		0	1	0	0	0	0	0	1	0	1	Rm			Rdn			x	x	x	x
Substract With Carry	SBCS	<RDn>,	<Rm>		0	1	0	0	0	0	0	1	1	0	Rm			Rdn			x	x	x	x
Rotate Right	RORS	<RDn>,	<Rm>		0	1	0	0	0	0	0	1	1	1	Rm			Rdn			x		x	x
Set Flags and bitwise AND	TST	<Rn>	<Rm>		0	1	0	0	0	0	1	0	0	0	Rm			Rn					x	x
Reverse Substruct from 0	RSBS	<Rd>,	<Rn>	#0	0	1	0	0	0	0	1	0	0	1	Rn->Rm			Rd			x		x	x
Compare registers	CMP	<Rn>	<Rm>		0	1	0	0	0	0	1	0	1	0	Rm			Rn			x	x	x	x
Compare negative	CMN	<Rn>	<Rm>		0	1	0	0	0	0	1	0	1	1	Rm			Rn			x	x	x	x
Logical OR	ORRS	<RDn>,	<Rm>		0	1	0	0	0	0	1	1	0	0	Rm			Rdn					x	x
Multiply Two Registers	MULS	<Rdm>	<Rn>	<Rdm>	0	1	0	0	0	0	1	1	0	1	Rn->Rm			Rdm					x	x
Bit Clear	BICS	<RDn>,	<Rm>		0	1	0	0	0	0	1	1	1	0	Rm			Rdn					x	x
Bitwise NOT	MVNS	<Rd>,	<Rm>		0	1	0	0	0	0	1	1	1	1	Rm			Rd					x	x

Tableau d’instructions

Load/Store					1	0	0	1	Opcode										
Store Register	STR	<Rt>	[SP , #<imm8>]		1	0	0	1	0	Rt			imm8						
Load Register	LDR	<Rt>	[SP , #<imm8>]		1	0	0	1	1	Rt			imm8						
Miscellaneaous 16-bit instrs					1	0	1	1	Opcode										
Add Immediate to SP	ADD	[SP ,]	SP	#<imm7>	1	0	1	1	0	0	0	0	0	imm7					
Substract Immediate from SP	SUB	[SP ,]	SP	#<imm7>	1	0	1	1	0	0	0	0	1	imm7					
Conditional Branch	**B**	**<label>**			1	1	0	1	Cond				imm8				**Check if Flags**		
Egalite	BEQ	<label>			1	1	0	1	0	0	0	0	imm8						1
Différence	BNE	<label>			1	1	0	1	0	0	0	1	imm8						0
reteenue	BCS	<label>			1	1	0	1	0	0	1	0	imm8		1				
pas de retenue	BCC	<label>			1	1	0	1	0	0	1	1	imm8		0				
négatif	BMI	<label>			1	1	0	1	0	1	0	0	imm8					1	
Positif ou nul	BPL	<label>			1	1	0	1	0	1	0	1	imm8					0	
dépassement de capacité	BVS	<label>			1	1	0	1	0	1	1	0	imm8			1			
Pas de dépassement de cpct	BVC	<label>			1	1	0	1	0	1	1	1	imm8			0			
Supérieur (non signé)	BHI	<label>			1	1	0	1	1	0	0	0	imm8			C == 1 && Z == 0			
Inferieur ou égal (non signé)	BLS	<label>			1	1	0	1	0	0	0	1	imm8			C == 0 && Z == 1			
supérieur ou égal (signé)	BGE	<label>			1	1	0	1	1	0	1	0	imm8			N == V			
inférieur (signé)	BLT	<label>			1	1	0	1	1	0	1	1	imm8			N != V			
supérieur (signé)	BGT	<label>			1	1	0	1	1	1	0	0	imm8			Z == 0 && N == V			
Inférieur ou égal (signé)	BLE	<label>			1	1	0	1	1	1	0	1	imm8			Z == 1 && Z != V			
toujours vrai	B ou BAL	<label>			1	1	0	1	1	1	1	0	imm8						

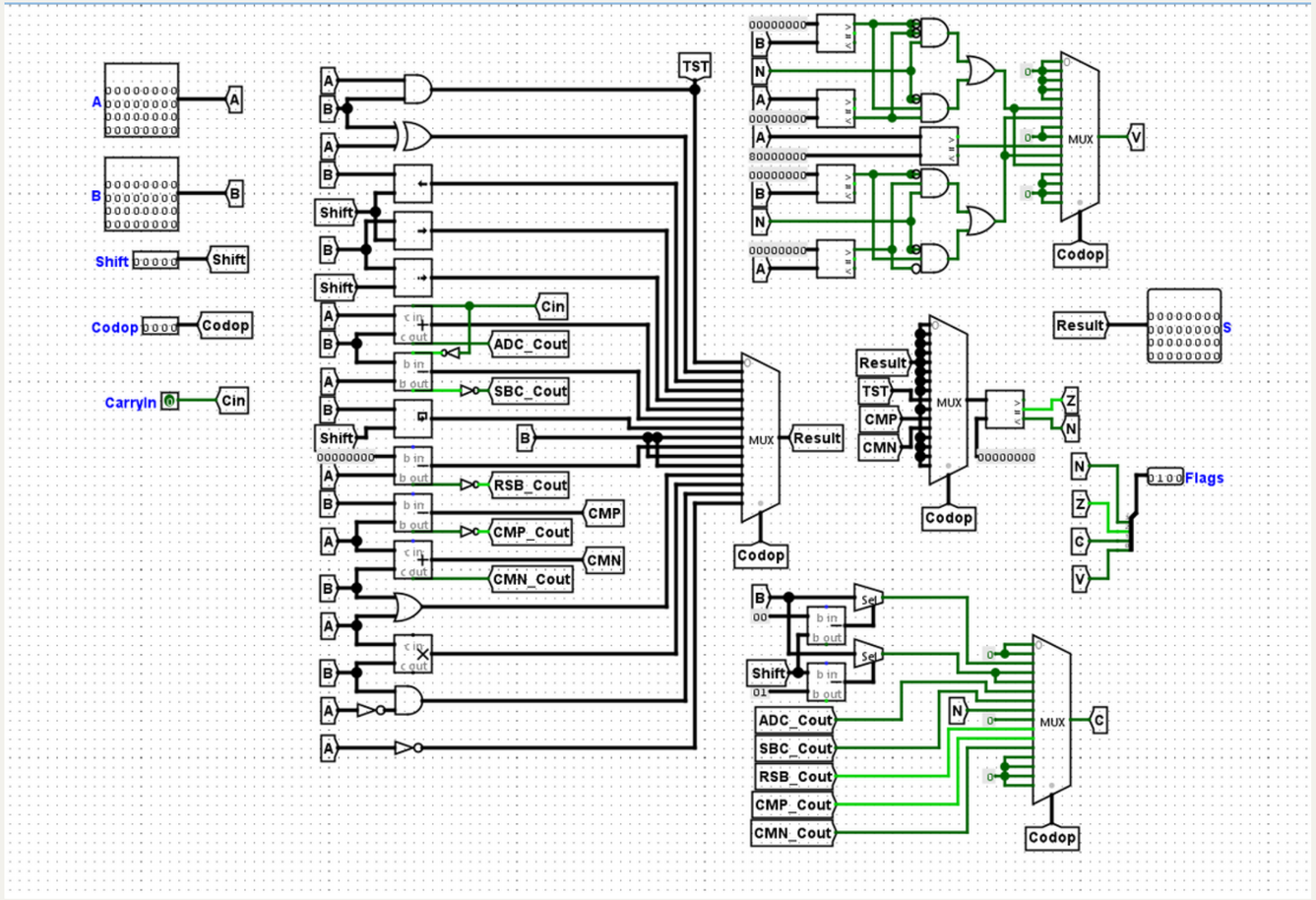
Tableau d’instructions

I.

Présentation des composants

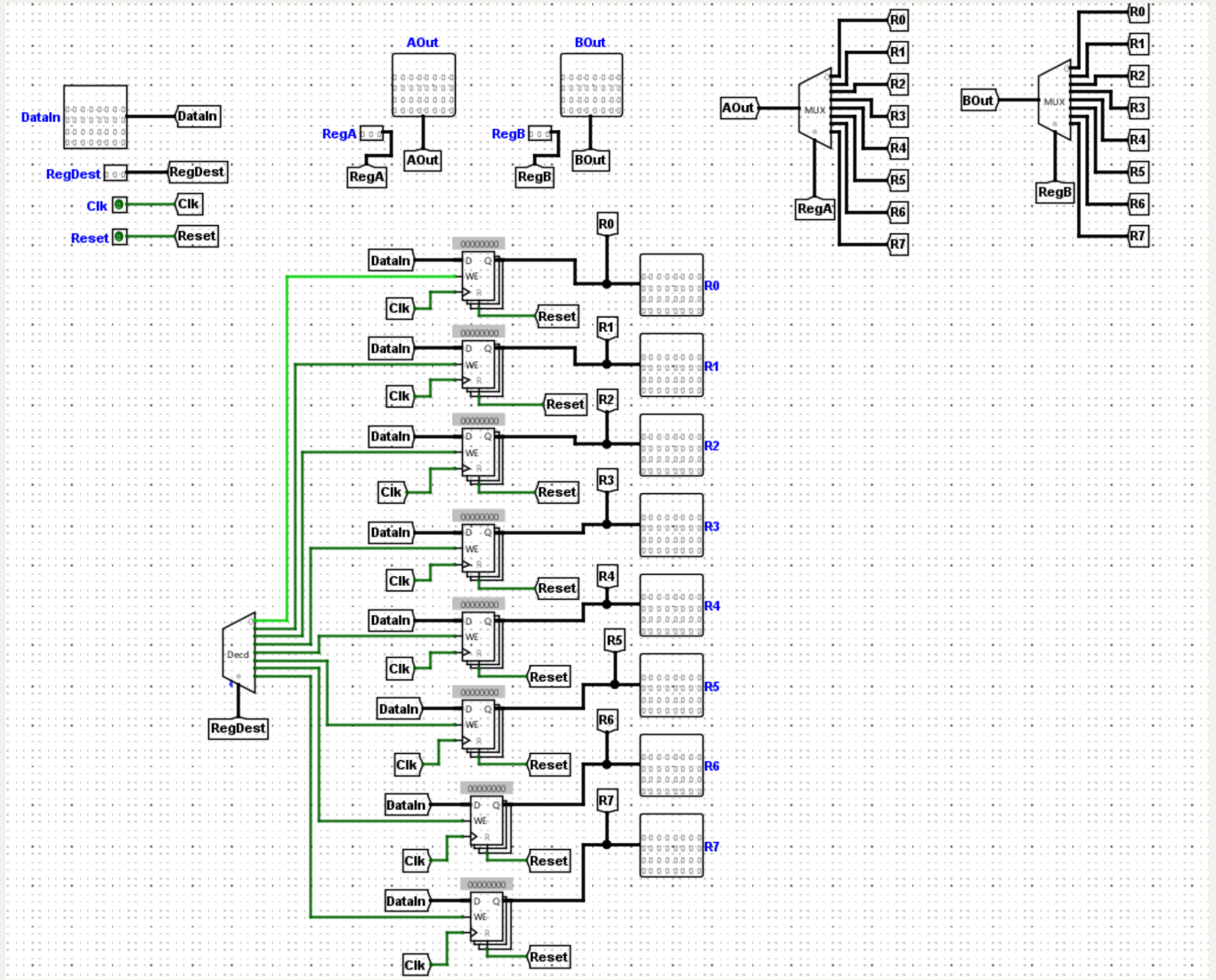
01. ALU

L’ALU se charge des calculs au sein du processeur avec la prise en charge des flags(drapeaux)



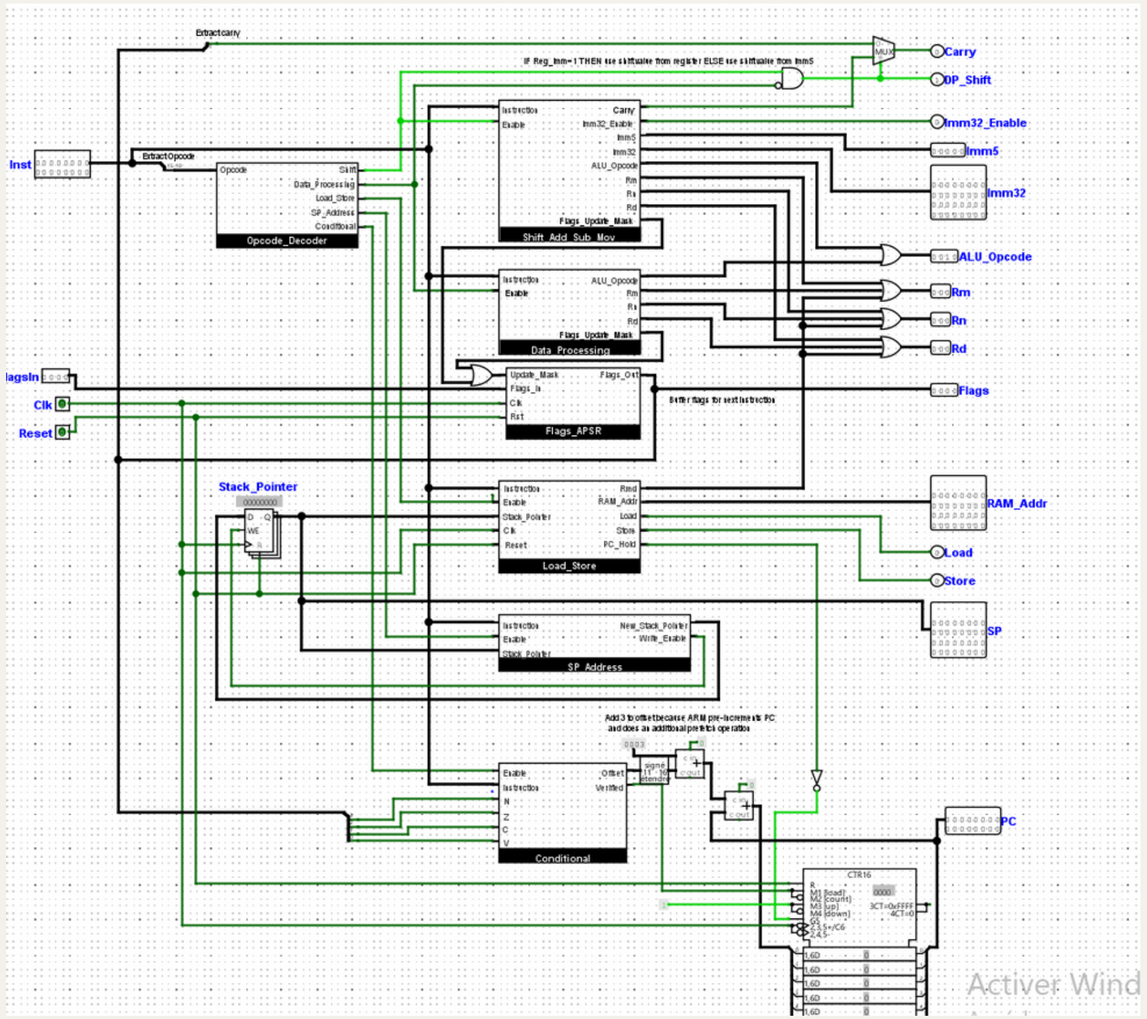
02. Banc de régistres

Il permet d’enregistrer les données avant et après l’opération



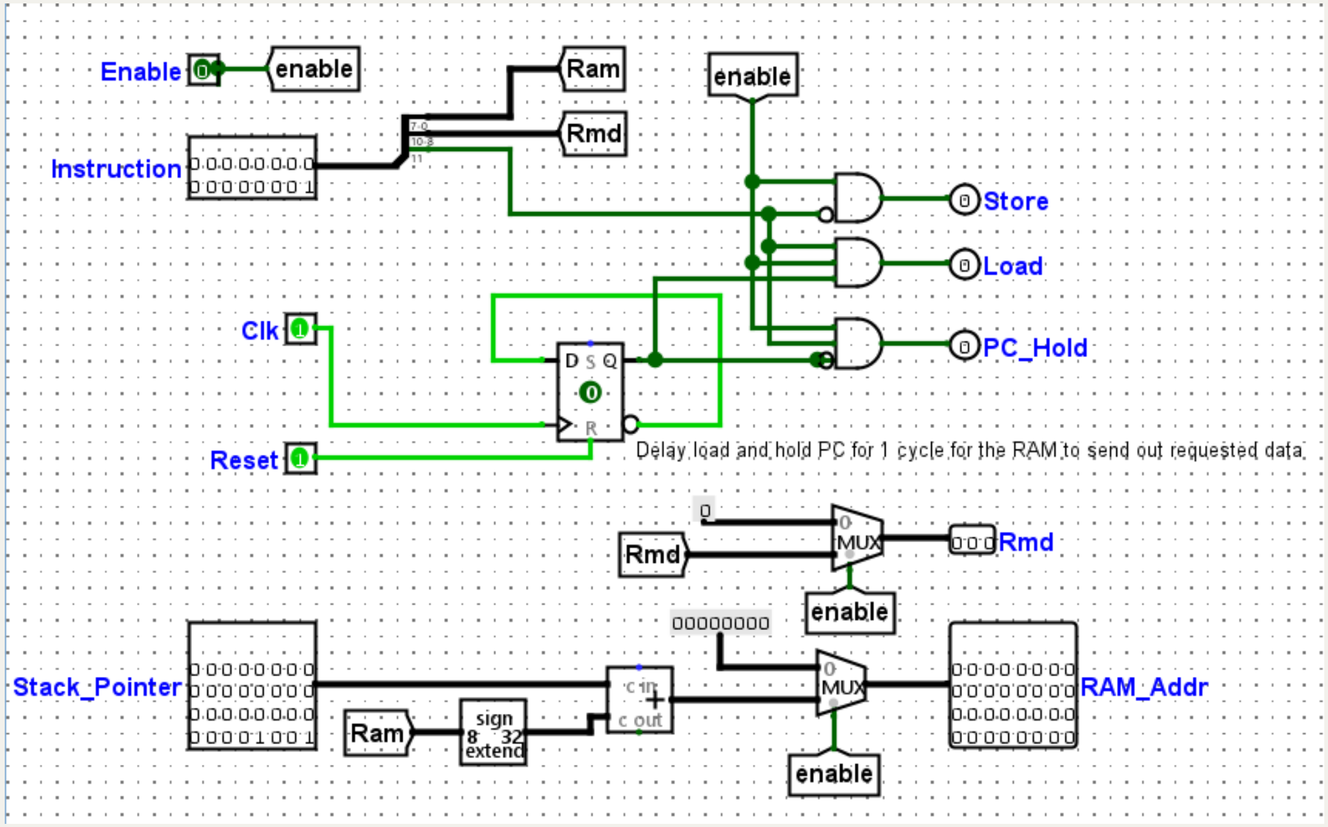
03. Controller

Il est en charge du décodage des instructions



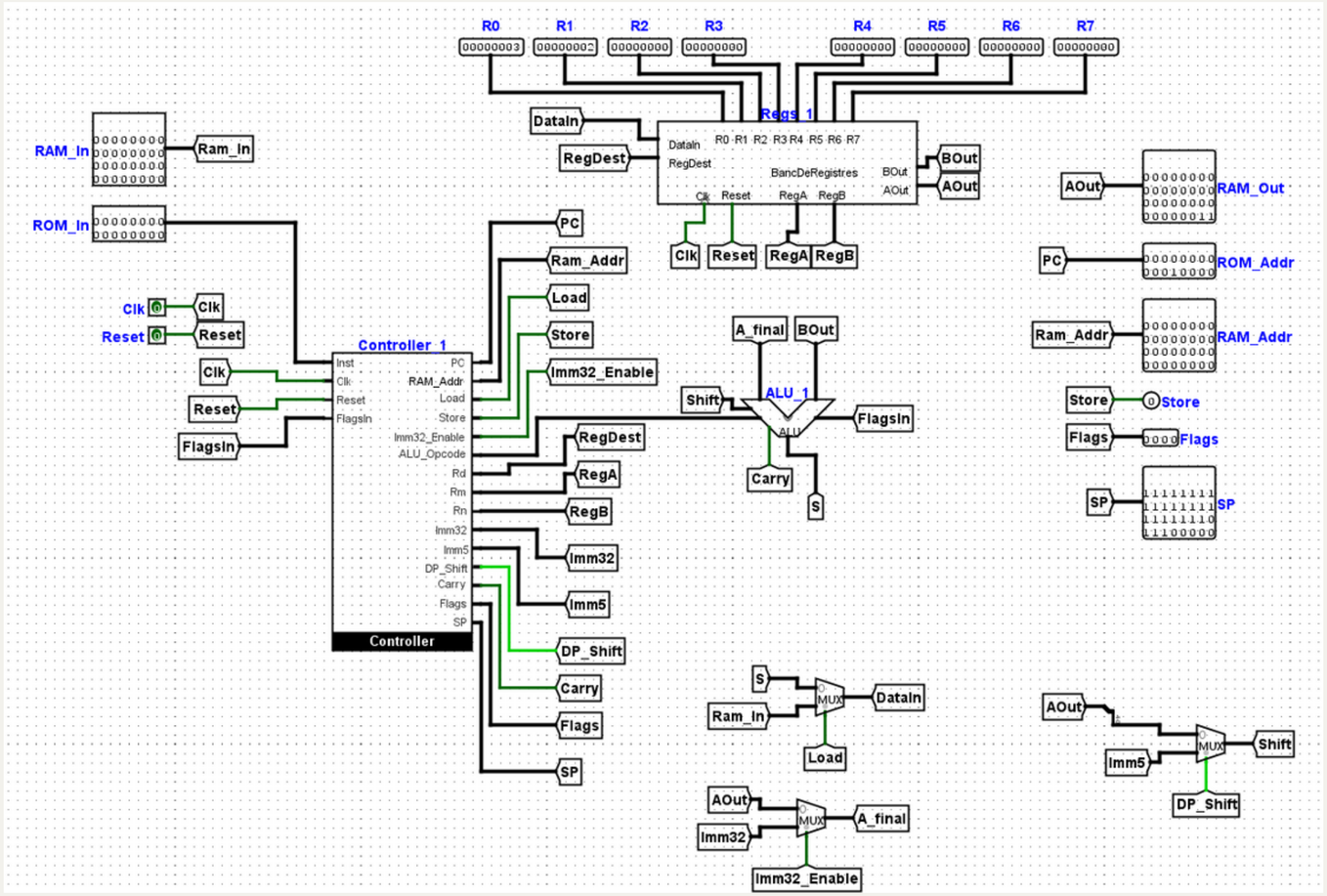
04. Load/Store

Il traite les instructions de lecture/écriture en mémoire



05. CPU

Il regroupe le contrôleur qui décode l’instruction, l’ALU qui effectue le calcul et le banc de registres qui fournit les données et enregistre le résultat.



06. Assembleur

Pourquoi Python?

Langage facile à utiliser

Syntaxe claire et lisible

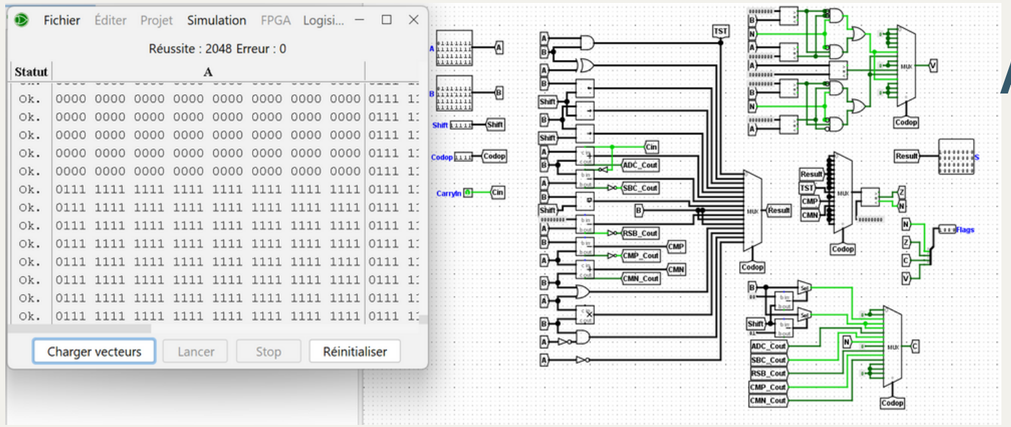
Peut être intégré à d’autres langages de programmation

Langage polyvalent pouvant être utilisé dans plusieurs domaines

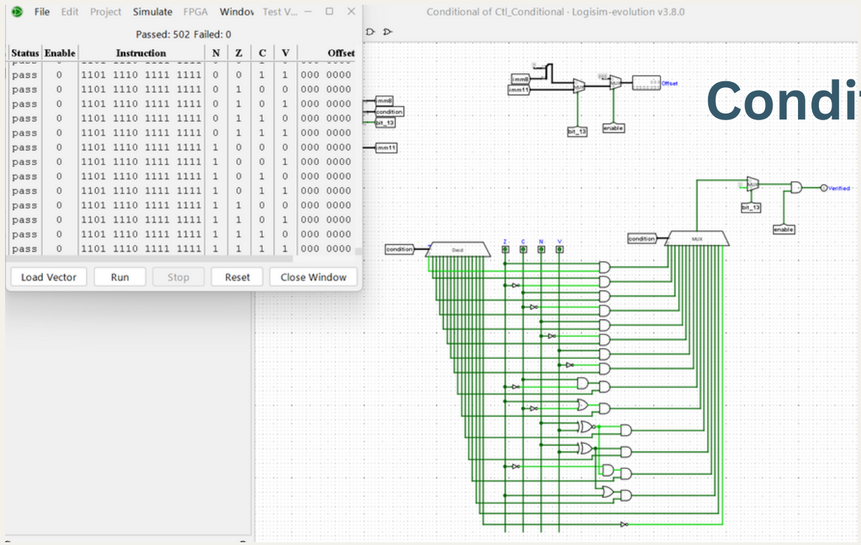
II.

Tests unitaires, assembleur, code C

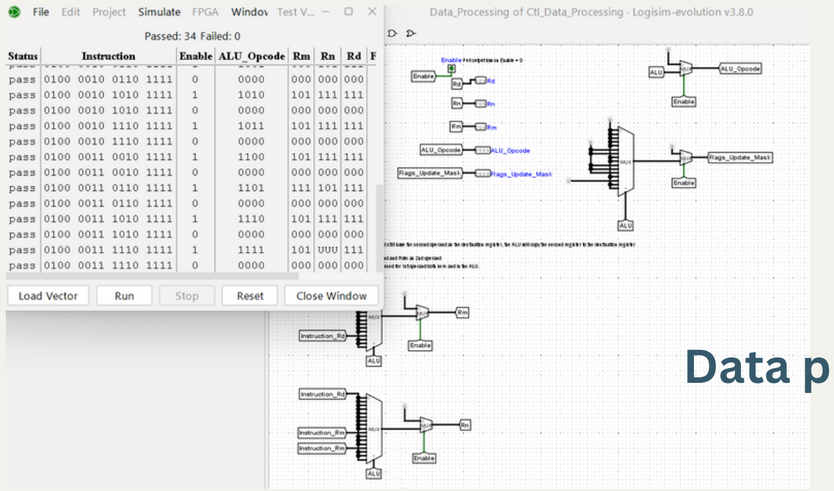
Tests unitaires



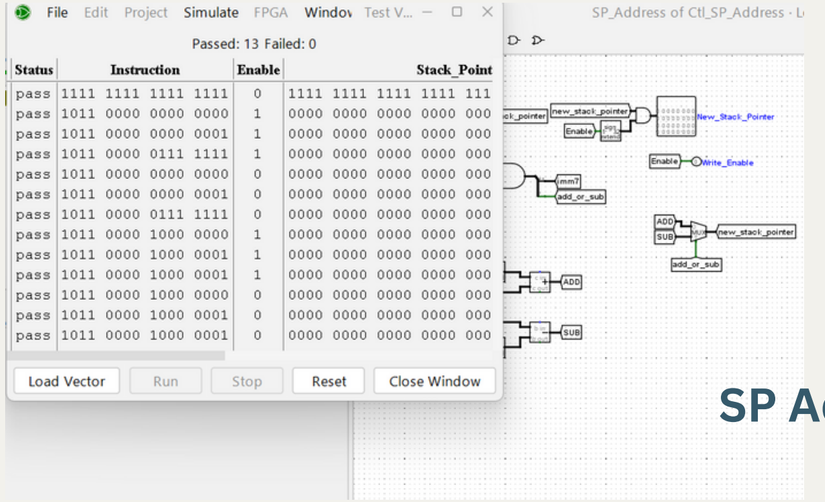
ALU



Conditional



Data processing



SP Adress

Tests unitaires

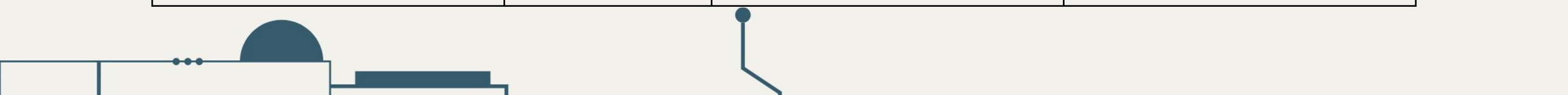
Tests	Réussites	Echecs
ALU	2048	0
Data Processing	34	0
Conditional	502	0
Opcode Decoder	29	0
Load & Store	32	0
Shift Add Sub Move	35	0
SP Adress	13	0
Controller	500	0
Register Bank	3	0
Taux de couverture	100%	0%

Tests assembleurs

Codes ASM	test passe	non testé
Conditional	1/1	0/1
DP_1_4	1/1	0/1
DP_5_10	1/1	0/1
DP_11_12	1/1	0/1
DP_13_16	1/1	0/1
Load_store	1/1	0/1
SP	1/1	0/1
SASM_1_4	1/1	0/1
SASM_5_8	1/1	0/1
taux de couverture	100%	0%

Tests Code C

Codes C	test asm passe	test logisim passe	non testé
calckeyb	1/1	1/1	0/1
calculator	1/1	1/1	0/1
simple_add	1/1	1/1	0/1
testfp	1/1	1/1	0/1
tty	1/1	1/1	0/1
my own test	1/1	1/1	0/1
taux de couverture	100%	100%	0%



III.
Démonstration simple_add

IV.
Analyse et Critique du Travail

Ce travail nous a permis de ...

Utiliser un éditeur de circuits numériques	Faire la conception materielle
Génerer du code binaire à partir d’un assembleur	Génerer du code binaire à partir du code C
Réaliser des tests et des simulations	S’organiser et travailler en tant qu’équipe

Quelques difficultés …

Spécifications difficilement déchiffrables pour certains composants

Tests Unitaires qui ne passent pas

Des limites dans les opérations de L’ALU

Faire tourner le LOAD&STORE

Les flags qu’on ne peut pas déterminer

V.
Travail d’équipe

Et le travail d'équipe?

Etapes de travail	Insuffisant	En cours	Bien	Terminé
Compréhension générale			●	
Outils utilisés				●
Mise en œuvre				●
Jeu d'instructions				●
Tests et validation				●

Séances	Pas commencé	Commencé mais difficultés rencontrées	En cours	Bien avancé	Terminé
ALU				●	
BdR					●
Mem			●		
Controler					●
ASM					●
Code C			●		
Tests					●
Integration				●	
Presentation	●				

Google Doc

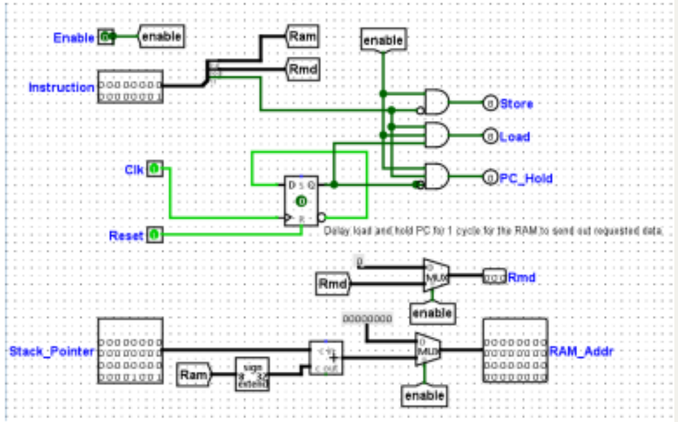
GitHub

Discord

Controller Load & Store (22/12/2023) [Sagesse]

Avancement :

Opérationnel, conforme et testé
Réalisé avec M. Litovsky et complété par Robin



Galère :

Déchiffrement des spécifications
Tests unitaires sans tour d'horloge

Séances	S1	S2	S3	S4	S5	S6
ALU	-	All	Sagesse	Done	Correction	Done
BdR	-	-	Stacy	Done	Done	Done
Mem	-	-	-	-	ALL	Done
Controler	-	-	Robin/ Sara	ALL	Done	Done
ASM	-	-	-	-	Robin	Done
Code C	-	-	-	-	Robin	Done
Tests	-	-	-	ALL	ALL	Done
Intégration	-	-	-	-	ALL	Done
Présentation	-	-	-	-	ALL	Done

Merci !!