PROJET PARM

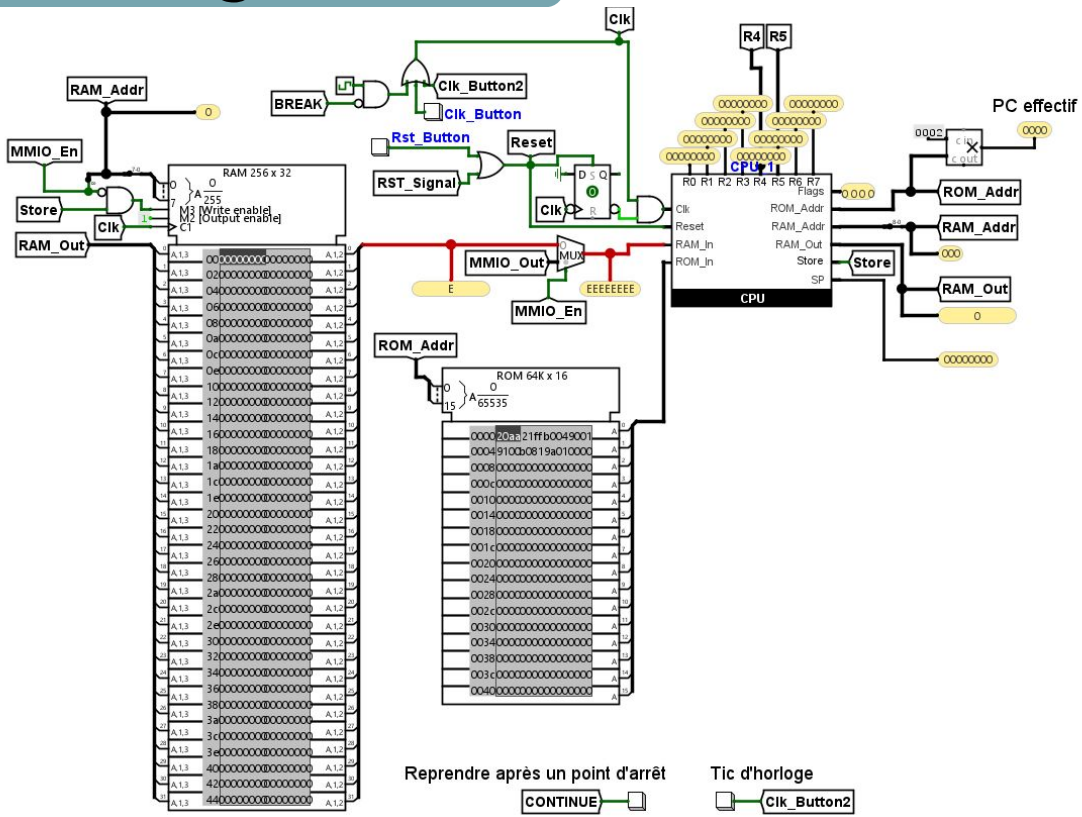
GROUPE : 24850 *HHKM*

Habib SIBILLE
Hannah CASABO
Kevin SUQUET
Marius FENNICHE

Introduction

Projet PARM : Création d’un processeur à l’aide de logisim.

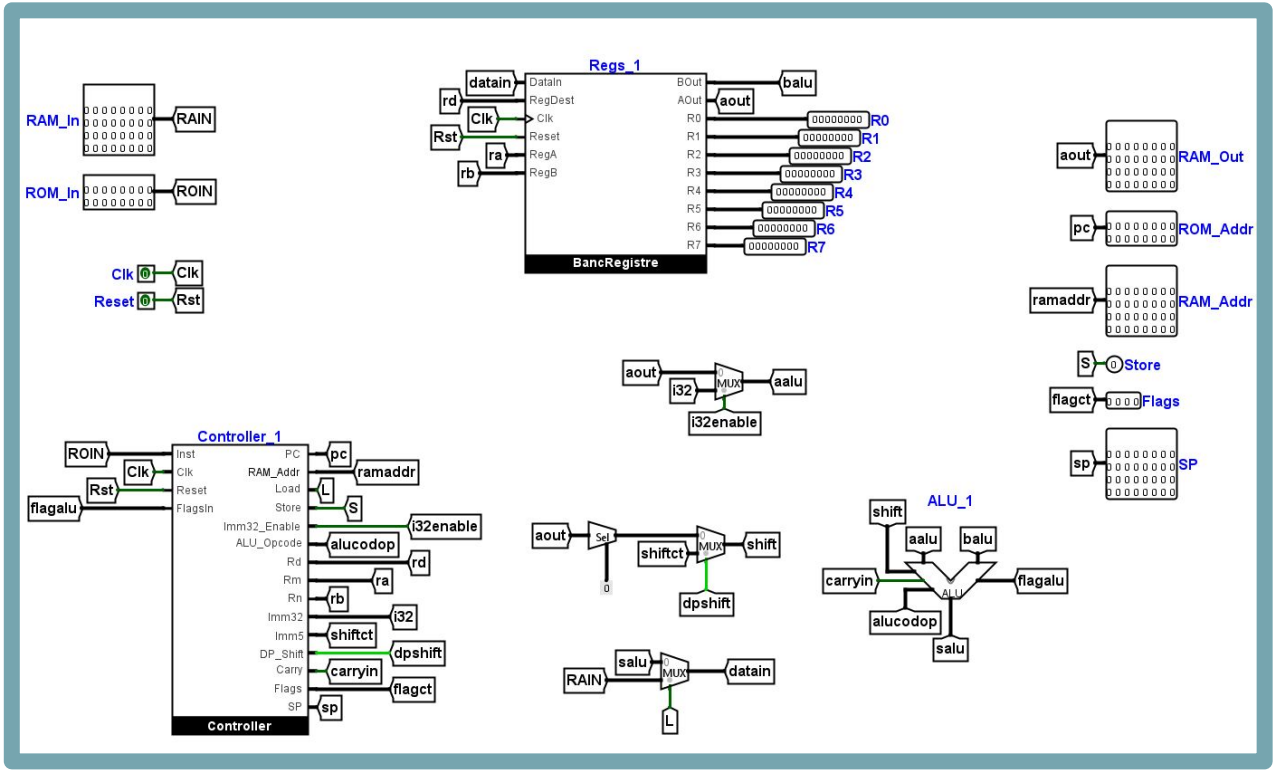
Machine Logisim



Le fichier Excel

Description	UAL Code				Bits															Flags			
	Instruction	Operandes			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	C	V	N	Z
Logical Shift Left	LSL$	<Rd>	<Rn>	#<imm5>	0	0	0	0	0	imm5					Rn			Rd			☑	☐	☑	☑
Logical Shift Right	LSR$	<Rd>	<Rn>	#<imm5>	0	0	0	0	1	imm5					Rn			Rd			☑	☐	☑	☑
Arithmetic Shift Right	ASR$	<Rd>	<Rn>	#<imm5>	0	0	0	1	0	imm5					Rn			Rd			☑	☐	☑	☑
Shift, add, sub, mov					0	0	opcode																	
Add register	ADD$	<Rd>	<Rn>	<Rm>	0	0	0	1	1	0	0	Rm			Rn			Rd			☑	☑	☑	☑
Subtract register	SUB$	<Rd>	<Rn>	<Rm>	0	0	0	1	1	0	1	Rm			Rn			Rd			☑	☑	☑	☑
Add immediate	ADD$	<Rd>	<Rn>	#<imm3>	0	0	0	1	1	1	0	Im3			Rn			Rd			☑	☑	☑	☑
Subtract immediate	SUB$	<Rd>	<Rn>	#<imm3>	0	0	0	1	1	1	1	Im3			Rn			Rd			☑	☑	☑	☑
Move	MOV$	<Rd>	#<imm8>		0	0	1	0	0	Rd		Imm8									☐	☐	☑	☑
Compare	CMP$	<Rd>	#<imm8>		0	0	1	0	1	Rd		Imm8									☐	☐	☑	☑
Add 8-bit immediate	ADD$	<Rd>	#<imm8>		0	0	1	1	0	Rdn		Imm8									☑	☑	☑	☑
Sub 8-bit immediate	SUB$	<Rd>	#<imm8>		0	0	1	1	1	Rdn		Imm8									☑	☑	☑	☑

CPU


RAM_In
RAIN
ROM_In
ROIN
Clk
Clk
Reset
Rst
Regs_1
datain
rd
Clk
Rst
ra
rb
DataIn
RegDest
Clk
Reset
RegA
RegB
BOut
AOut
R0
R1
R2
R3
R4
R5
R6
R7
balu
aout
R0
R1
R2
R3
R4
R5
R6
R7
BancRegistre
aout
RAM_Out
pc
ROM_Addr
ramaddr
RAM_Addr
S
Store
flagct
Flags
sp
SP
aout
i32
MUX
aalu
i32enable
Controller_1
ROIN
Clk
Rst
flagalu
Inst
Clk
Reset
FlagsIn
PC
RAM_Addr
Load
Store
Imm32_Enable
ALU_Opcode
Rd
Rm
Rn
Imm32
Imm5
DP_Shift
Carry
Flags
SP
pc
ramaddr
L
S
i32enable
alucodop
rd
ra
rb
i32
shiftct
dpshift
carryin
flagct
sp
Controller
aout
Sel
0
shiftct
MUX
shift
dpshift
RAIN
salu
MUX
datain
L
ALU_1
shift
aalu
balu
carryin
ALU
flagalu
alucodop
salu

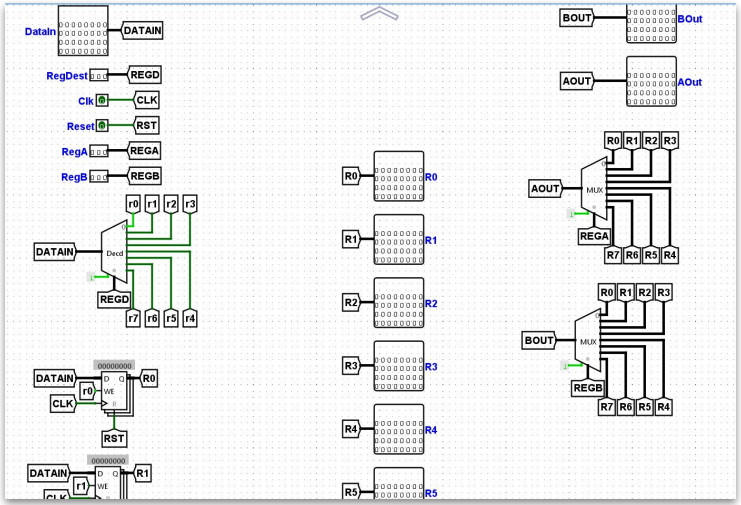
banc de registre

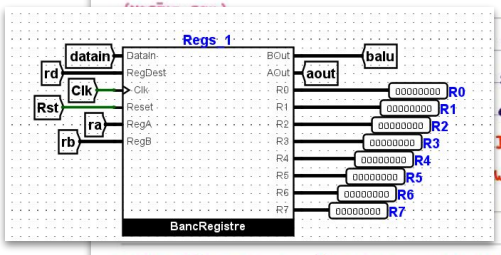
Le banc de registre

Comprendre le rôle du banc de registre :

-> Concept de registre

->mémoire rapide





ALU

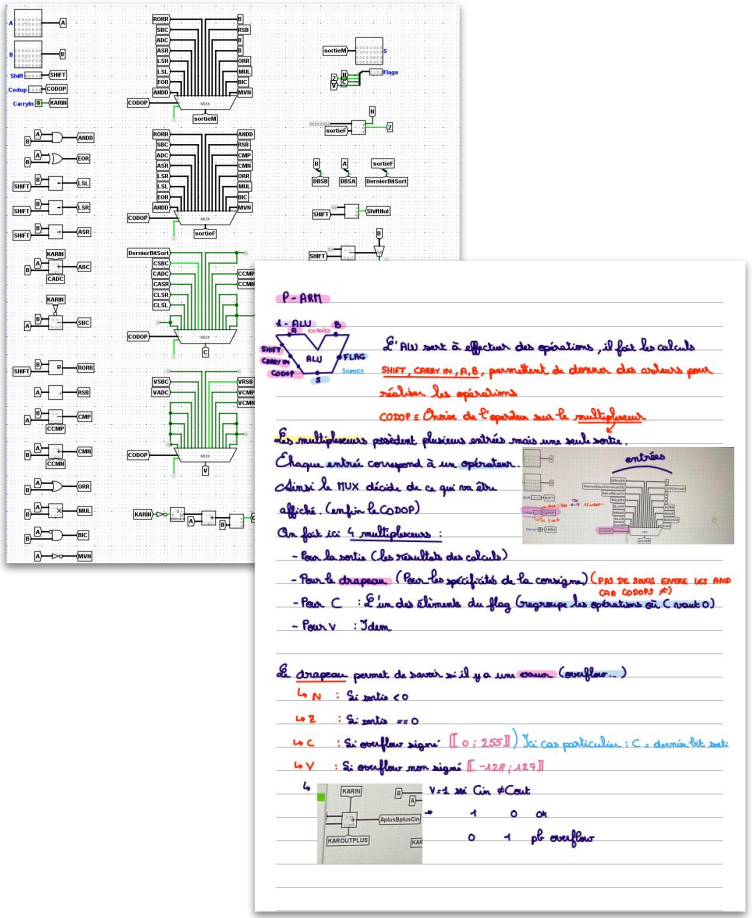
Comprendre le fonctionnement de l'ALU :

-> Opérations sur des entiers

-> Les flags

-> Le comportement

Le réaliser, comprendre son lien et ses interactions avec d'autres composants

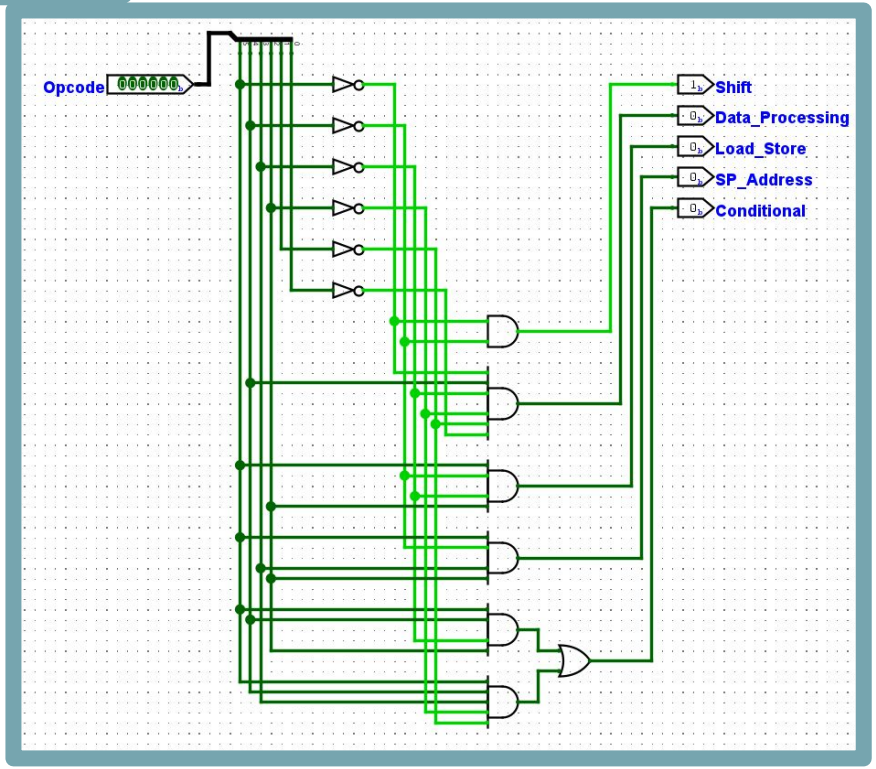


Tests ALU

```
#Association Entrées/Sorties à leur nombre de bits respectifs.

A[32] B[32] Shift[5] Codop[4] CarryIn[1] S[32] Flags[4]

#List of outputs
#A B Shift Codop CarryIn S Flags

#Test "and"

11111111111111111111111111111111 10111111111111111111111111111100 xxxxx 0000 x 10111111111111111111111111111100 1000
11111111111111111111111111111111 00000000000000000000000000000000 xxxxx 0000 x 00000000000000000000000000000000 0100

#Test "xor"

00001111111111111111111111111111 11111111111111111111111111111111 xxxxx 0001 x 11110000000000000000000000000000 1000
11111111111111111111111111111111 11111111111111111111111111111111 xxxxx 0001 x 00000000000000000000000000000000 0100

#Test "B << Shift"(BL)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00100000000000000000000000000000 00010 0010 x 10000000000000000000000000000000 1000
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00100000000000000000000000000000 00011 0010 x 00000000000000000000000000000000 0110

#Test "B >> Shift"(BR)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 10000000000000000000000000000000 00000 0011 x 10000000000000000000000000000000 1000
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00000000000000000000000000000001 00001 0011 x 00000000000000000000000000000000 0110

#Test "B >> Shift arith"

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00100000000000000000000000000001 00001 0100 x 00010000000000000000000000000000 0010
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 10000000000000000000000000000000 00010 0100 x 11100000000000000000000000000000 1000
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00000000000000000000000000000001 00001 0100 x 00000000000000000000000000000000 0110

#Test "A+B+CarryIn"

00000000000000000000000000000001 00000000000000001111111111111111 xxxxx 0101 0 00000000000000010000000000000000 0000
11111111111111111111111111111111 00000000000000000000000000000001 xxxxx 0101 0 00000000000000000000000000000000 0110
10000000000000000000000000000000 10000000000000000000000000000001 xxxxx 0101 1 00000000000000000000000000000010 0011
01000000000000000000000000000000 01000000000000000000000000000000 xxxxx 0101 1 10000000000000000000000000000001 1001
00000000000000000000000000000000 11111111111111111111111111111111 xxxxx 0101 1 00000000000000000000000000000000 0110
```

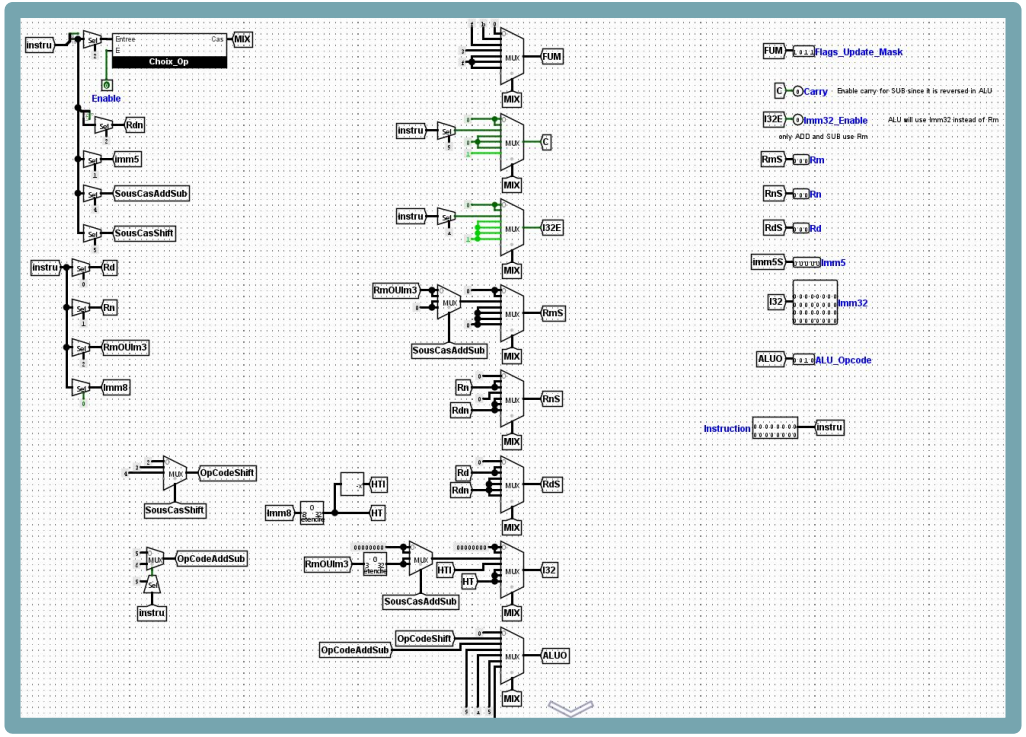
Les composantes

Décodeur d'instruction



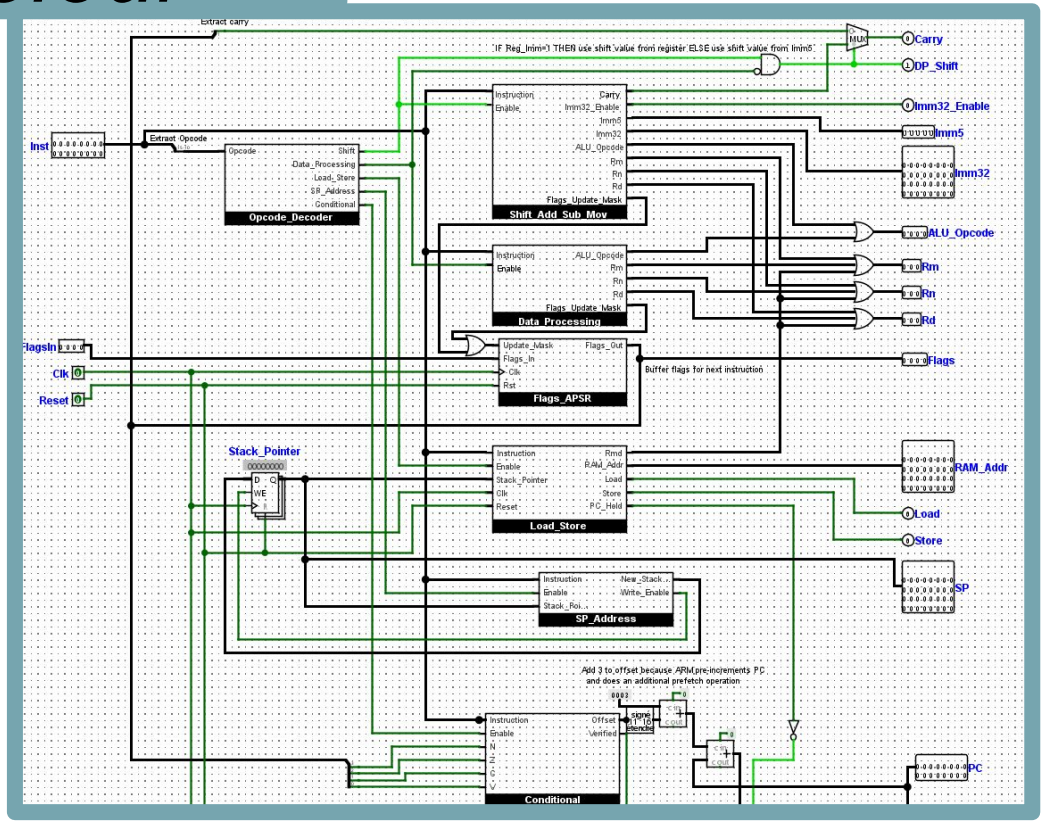
Les composantes

Shit.add.sub.mov



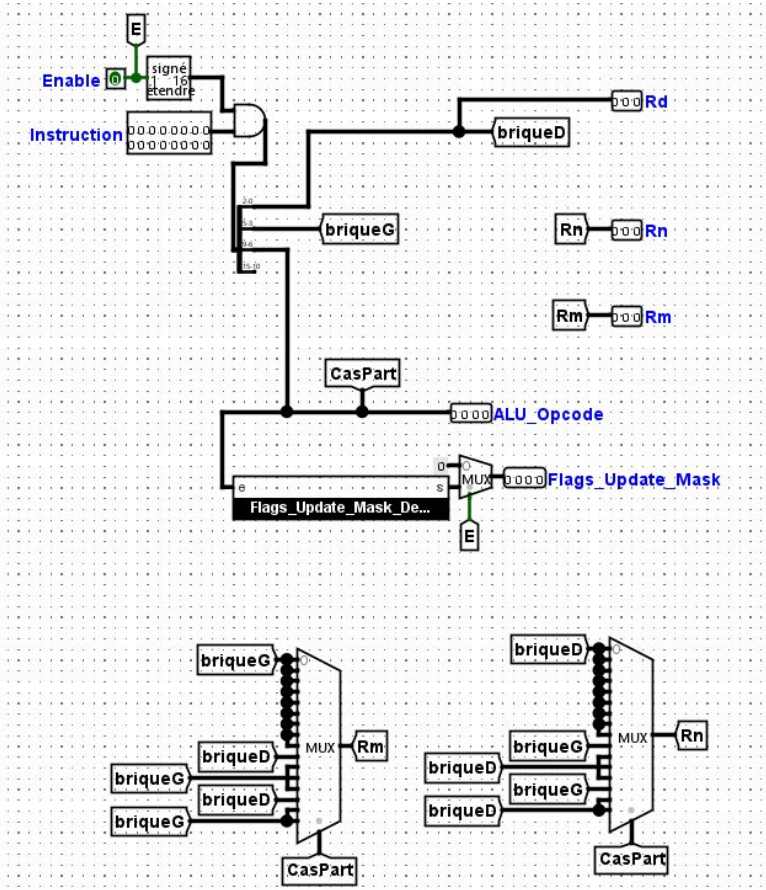
Le contrôleur

Contrôleur



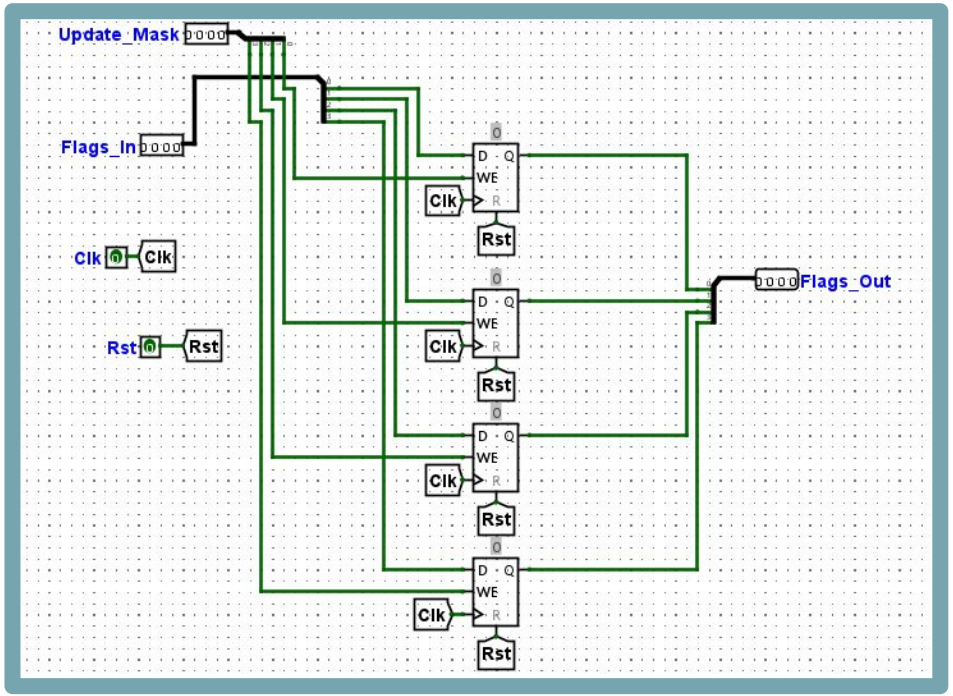
Sous composants

Data Processing



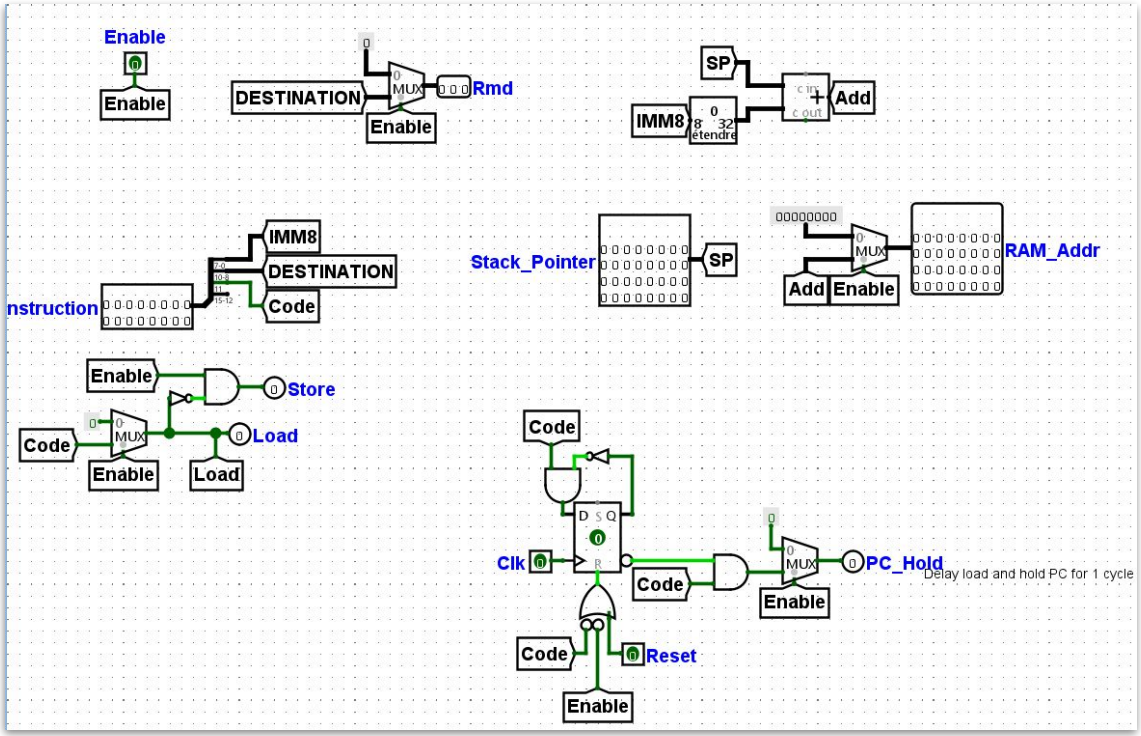
Sous composants

Flags APSR



Sous composants

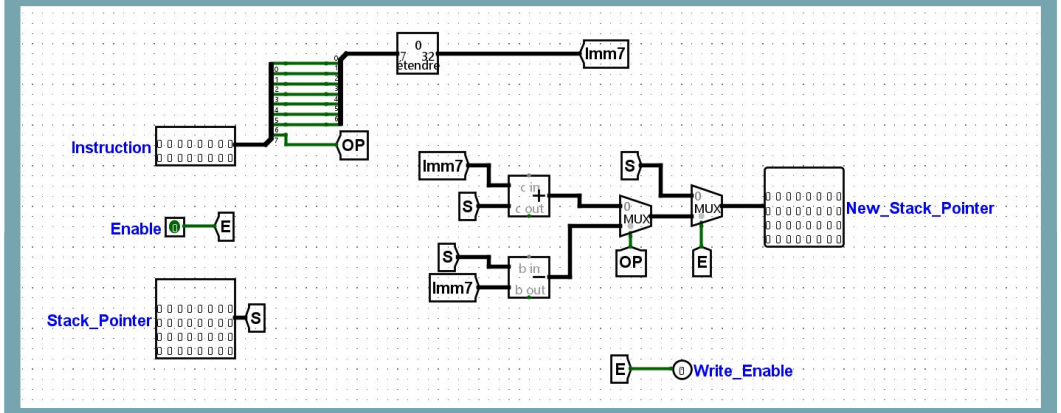
Load/Store



Sous composants

SP Address

- Mise à jour du pointeur de pile
- Addition = 0
- Soustraction = 1
- Disable => initial SP
- Hésitation sur le décalage

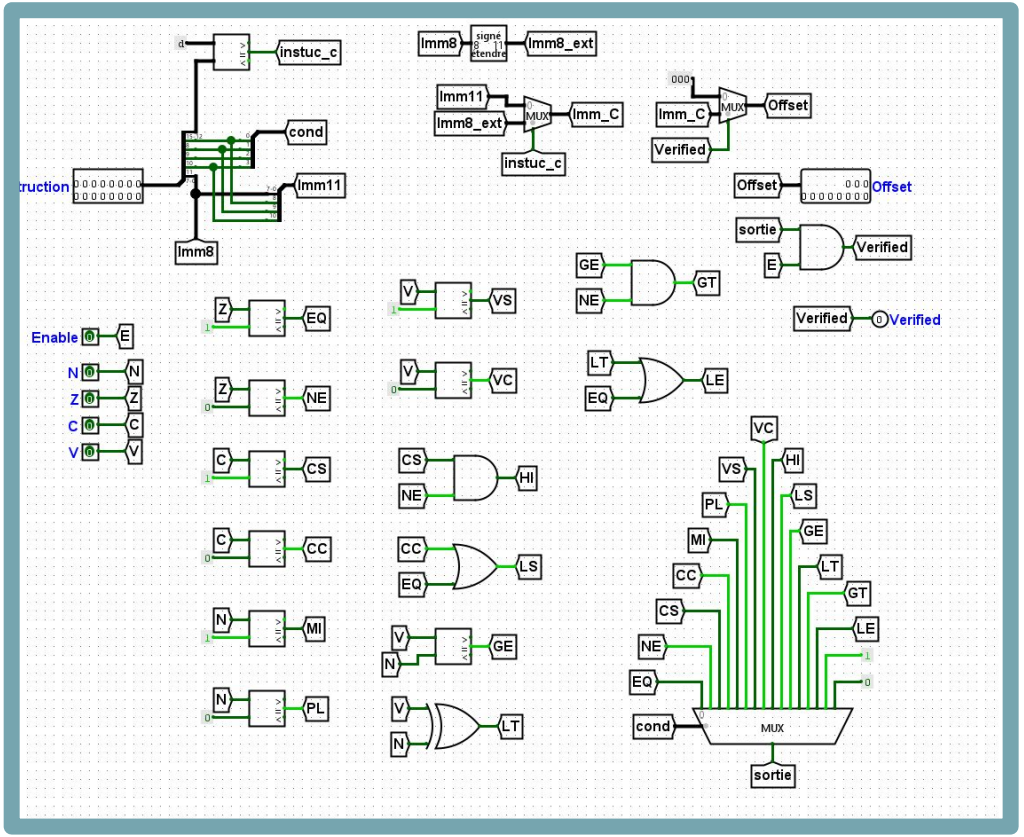



```
# ADD SP, #0   (imm7 = 0 → +0)
1011000000000000 1 00000000000000000000000000001000 1 00000000000000000000000000001000

# ADD SP, #4   (imm7 = 1 → +4)
1011000000000001 1 00000000000000000000000000001000 1 00000000000000000000000000001100

# ADD SP, #16 (imm7=4 -> +16)
1011000000000100 1 00000000000000000000000000010000 1 00000000000000000000000000100000

# ADD SP, #28  (imm7 = 7 → +28)
1011000000000111 1 00000000000000000000000000100000 1 00000000000000000000000000111100
```

Sous composants

Conditional

- Vérification des conditions
- Comparaison avec le Flag
- Une condition = une possibilité
- 16 - 1 Unconditional - 1 True



Tests & Assembleurs

Demo

Conclusion