

UnLitPourSimonEtThéoSVP

Projet PARM 2022



MAUROIS Quentin
FROMENT Lorenzo
DELIAS Théo
BEUREL Simon

Sommaire

- Présentation du Compilateur
- Présentation du travail sur Logisim
- Tests
- Points à améliorer pour le futur

Compilateur : Le langage Assembleur

```
void run(void){  
    int a,b,c;  
    a = 0;  
    b = 1;  
    c = a+b;  
}
```



Génération
du code
assembleur



```
run:  
    .fnstart  
    .pad    #12  
    sub sp, #12  
    movs    r0, #0  
    str r0, [sp, #8]  
    movs    r0, #1  
    str r0, [sp, #4]  
    ldr r0, [sp, #8]  
    ldr r1, [sp, #4]  
    adds    r0, r0, r1  
    str r0, [sp]  
    add sp, #12  
    bx  lr
```

*“clang -S -target arm -none -eabi
-mcpu=cortex -m0 -O0 -mthumb
-nostdlib -I./ include main.c” :*

Compilateur : Les instructions

Description	Instruction	UAL code				Bits																Flags							
		operands				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	C	V	N	Z				
Shift, add, sub, mov						0	0	opcode																					
Logical Shift Left	LSLS	<Rd>	<Rm>	#<imm5>		0	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				
Add register	ADDS	<Rd>	<Rn>	<Rm>		0	0	0	1	1	0	0	Rm					Rn		Rd		X	X	X	X				
Subtract 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				
Subtract 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																													
Bitwise AND	ANDS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn				X	X	X				
Exclusive OR	EORS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn				X	X	X				
Logical Shift Left	LSLS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X		X	X	X				
Logical Shift Right	LSRS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X		X	X	X				
Arithmetic Shift Right	ASRS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X		X	X	X				
Add with Carry	ADCS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X	X	X	X					
Subtract with Carry	SBCS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X	X	X	X					
Rotate Right	RORS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn		X		X	X	X				
Set Flags on bitwise AND	TST	<Rn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rn				X	X	X				
Reverse Subtract from 0	RSBS	<Rd>	<Rn>	#0		0	1	0	0	0	0	opcode					Rn		Rd				X	X	X				
Compare Registers	CMP	<Rn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rn		X	X	X	X					
Compare Negative	CMN	<Rn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rn		X	X	X	X					
Logical OR	ORRS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn				X	X	X				
Multiply Two Registers	MULS	<Rdm>	<Rm>	<Rdm>		0	1	0	0	0	0	opcode					Rn		Rdm				X	X	X				
Bit Clear	BICS	<Rdn>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rdn				X	X	X				
Bitwise NOT	MVNS	<Rd>	<Rm>			0	1	0	0	0	0	opcode					Rm		Rd				X	X	X				
Load / Store																													
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												
Miscellaneous																													
Add Immediate to SP	ADD	[SP]	SP	#<imm7>		1	0	1	1	0	0	0	0	0	imm7														
Subtract Immediate from SP	SUB	[SP]	SP	#<imm7>		1	0	1	1	0	0	0	0	1	imm7														
Conditional Branch	B<c>	<label>				1	1	0	1	opcode cond											imm8								
Equal	EQ	<label>				1	1	0	1	0	0	0	0	0	imm8														
Not Equal	NE	<label>				1	1	0	1	0	0	0	0	1	imm8										0				
Carry set	CS	<label>				1	1	0	1	0	0	0	1	0	imm8										1				
Carry clear	CC	<label>				1	1	0	1	0	0	1	1	0	imm8										0				
Minus, negative	MI	<label>				1	1	0	1	0	1	1	0	0	imm8										1				
Plus, positive or zero	PL	<label>				1	1	0	1	0	1	0	1	0	imm8										0				
Overflow	VS	<label>				1	1	0	1	0	1	1	0	0	imm8										1				
No overflow	VC	<label>				1	1	0	1	0	1	1	1	0	imm8										0				
Unsigned higher	HI	<label>				1	1	0	1	1	0	0	0	0	imm8										C==1 and Z==0				
Unsigned lower or same	LS	<label>				1	1	0	1	1	0	0	1	0	imm8										C==0 or Z==1				
Signed greater than or equal	GE	<label>				1	1	0	1	1	0	1	0	0	imm8										N==V				
Signed less than	LT	<label>				1	1	0	1	1	0	1	1	0	imm8										N!=V				
Signed greater than	GT	<label>				1	1	0	1	1	1	0	0	0	imm8										Z==0 and N==V				
Signed less than or equal	LE	<label>				1	1	0	1	1	1	0	1	0	imm8										Z==1 or N!=V				
Always (unconditional)	none or AL	<label>				1	1	0	1	1	1	1	0	0	imm8														

Shift, add, sub, mov

Shift, add, sub, mov					0	0	opcode											
Logical Shift Left	LSLS	<Rd>	<Rm>	#<imm5>	0	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	
Add register	ADDS	<Rd>	<Rn>	<Rm>	0	0	0	1	1	0	0	Rm	Rn	Rd	X	X	X X	
Subtract 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	
Subtract 3-bit immediate	SUBS	<Rd>	<Rn>	#<imm3>	0	0	0	1	1	1	1	imm3	Rn	Rd	X	X	X X	
Move	MOV	<Rd>	#<imm8>		0	0	1	0	0			Rd	imm8				X X	

Compilateur : Data Processing

Data Processing					0	1	0	0	0	0	opcode						C	V	N	Z
Bitwise AND	ANDS	<Rdn>	<Rm>		0	1	0	0	0	0	0	0	0	0	Rm	Rdn			X	X
Exclusive 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	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
Subtract 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 on bitwise AND	TST	<Rn>	<Rm>		0	1	0	0	0	0	1	0	0	0	Rm	Rn			X	X
Reverse Subtract from 0	RSBS	<Rd>	<Rn>	#0	0	1	0	0	0	0	1	0	0	1	Rn	Rd			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	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

Load / Store & Miscellaneous

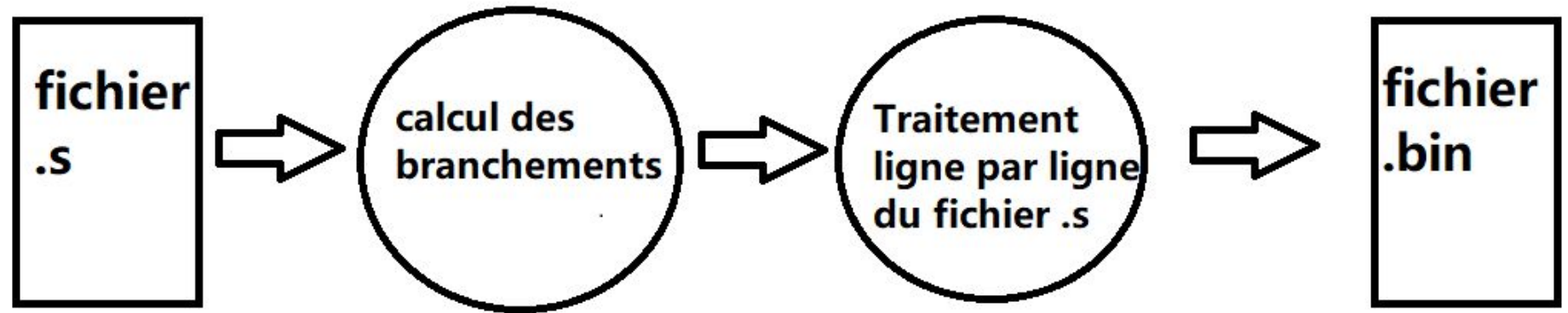
Load / Store										1	0	0	1		C	V	N	Z
Store Register	STR	<Rt>	[SP, #imm8]	1	0	0	1	0	Rt	imm8								
Load Register	LDR	<Rt>	[SP, #imm8]	1	0	0	1	1	Rt	imm8								

Miscellaneous					1	0	1	1	opcode							C	V	N	Z
Add Immediate to SP	ADD	[SP]	SP	#<imm7>	1	0	1	1	0	0	0	0	0	imm7					
Subtract Immediate from SP	SUB	[SP]	SP	#<imm7>	1	0	1	1	0	0	0	0	1	imm7					

Conditional & Unconditional Branch

Conditional Branch	B<> c	<label>			1	1	0	1	opcode cond				imm8	C	V	N	Z
Equal	EQ	<label>			1	1	0	1	0	0	0	0	imm8				1
Not Equal	NE	<label>			1	1	0	1	0	0	0	1	imm8				0
Carry set	CS	<label>			1	1	0	1	0	0	1	0	imm8	1			
Carry clear	CC	<label>			1	1	0	1	0	0	1	1	imm8	0			
Minus, negative	MI	<label>			1	1	0	1	0	1	0	0	imm8			1	
Plus, positive or zero	PL	<label>			1	1	0	1	0	1	0	1	imm8			0	
Overflow	VS	<label>			1	1	0	1	0	1	1	0	imm8		1		
No overflow	VC	<label>			1	1	0	1	0	1	1	1	imm8		0		
Unsigned higher	HI	<label>			1	1	0	1	1	0	0	0	imm8	C==1 and Z==0			
Unsigned lower or same	LS	<label>			1	1	0	1	1	0	0	1	imm8	C==0 or Z==1			
Signed greater than or equal	GE	<label>			1	1	0	1	1	0	1	0	imm8	N==V			
Signed less than	LT	<label>			1	1	0	1	1	0	1	1	imm8	N!=V			
Signed greater than	GT	<label>			1	1	0	1	1	1	0	0	imm8	Z==0 and N==V			
Signed less than or equal	LE	<label>			1	1	0	1	1	1	0	1	imm8	Z==1 or N!=V			
Always (unconditional)	none or AL	<label>			1	1	0	1	1	1	1	0	imm8				

Compilateur : Process d'exécution

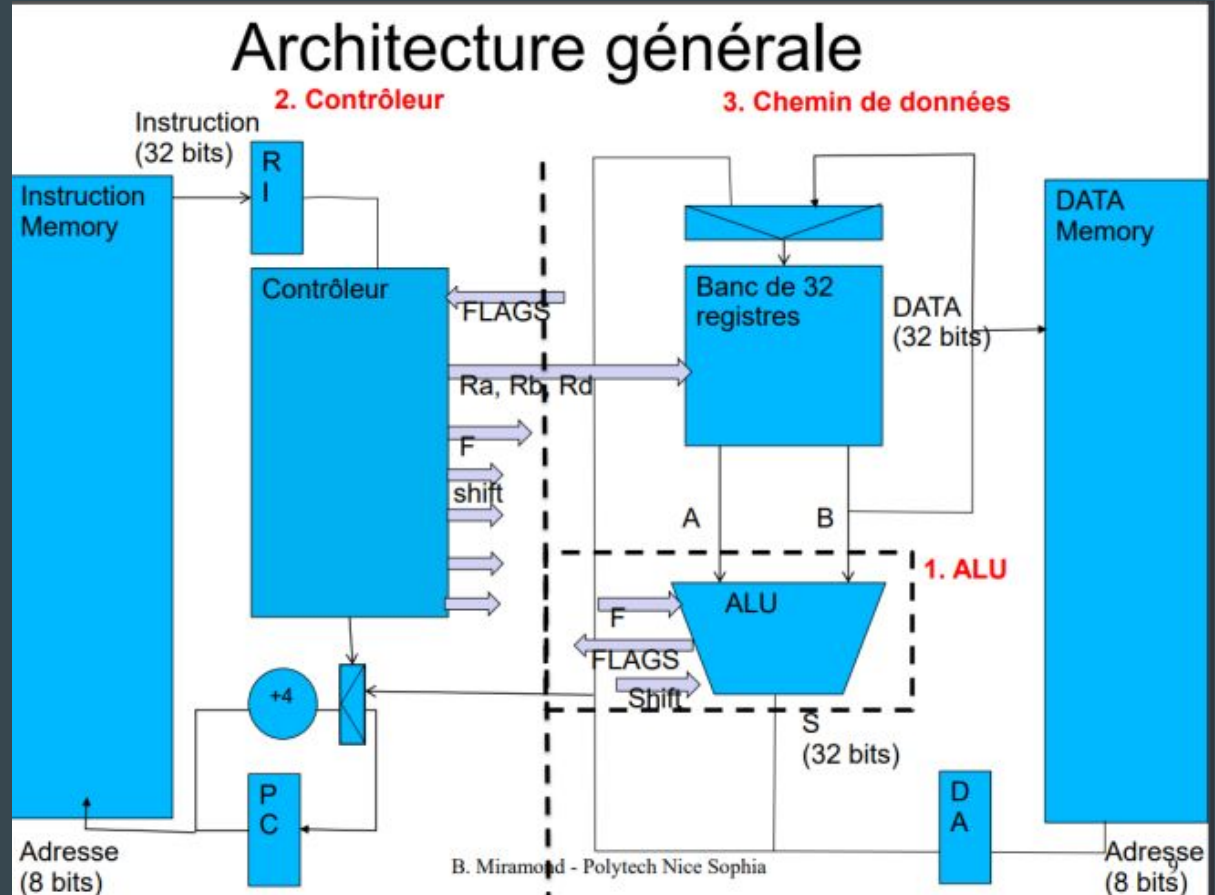


Compilateur : Code

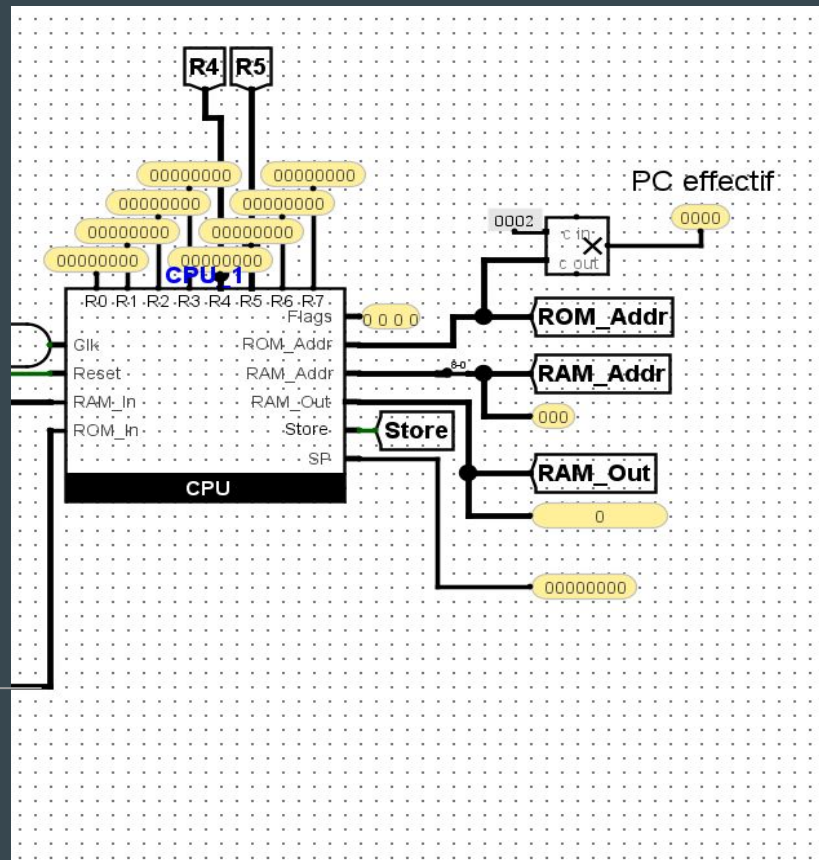
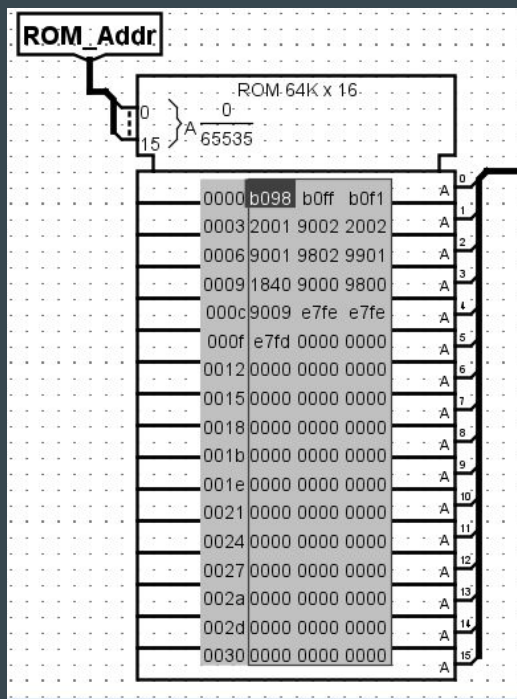
- Programmation en langage Java
- Utilisation de 3 classes :
 - Main
 - Compiler
 - Engine

Engine	Compiler	Main
Engine()	Compiler()	Main()
generateLexa(String)	process(String) void	main(String[]) void
STR(String)		
RORS(String)		
ORRS(String)		
EQ(String, HashMap<String, Integer>)		
RSBS(String)		
ASRS(String)		
ADDS_Register(String)		
LDR(String)		
CMP(String)		
MVNS(String)		
AL(String, HashMap<String, Integer>)		
SUB(String)		
LSRS(String)		
VC(String, HashMap<String, Integer>)		
ADCS(String)		
CC(String, HashMap<String, Integer>)		
generateBinaryFromString(int, int)		
NE(String, HashMap<String, Integer>)		
PL(String, HashMap<String, Integer>)		
LE(String, HashMap<String, Integer>)		
MULS(String)		
GE(String, HashMap<String, Integer>)		
EORS(String)		
VS(String, HashMap<String, Integer>)		
SUBS_Register(String)		
generateRegister(String)		
generateMMMax4(String, int)		
UB(String)		
scanBranch(FileInputStream) HashMap<String, Integer>		
LSLS(String)		
BICS(String)		
CMN(String)		
MI(String, HashMap<String, Integer>)		
HI(String, HashMap<String, Integer>)		
ADD(String)		
ASRS_data_processing(String)		
SBCS(String)		
B(String, HashMap<String, Integer>)		
MOVN(String)		
generateMMN(String, int)		
LT(String, HashMap<String, Integer>)		
CS(String, HashMap<String, Integer>)		
println(String)		
LSLS_data_processing(String)		
SUBS_3bits(String)		
LS(String, HashMap<String, Integer>)		
LSRS_data_processing(String)		
ADDS_3bits(String)		
TST(String)		
ANDS(String)		
GT(String, HashMap<String, Integer>)		

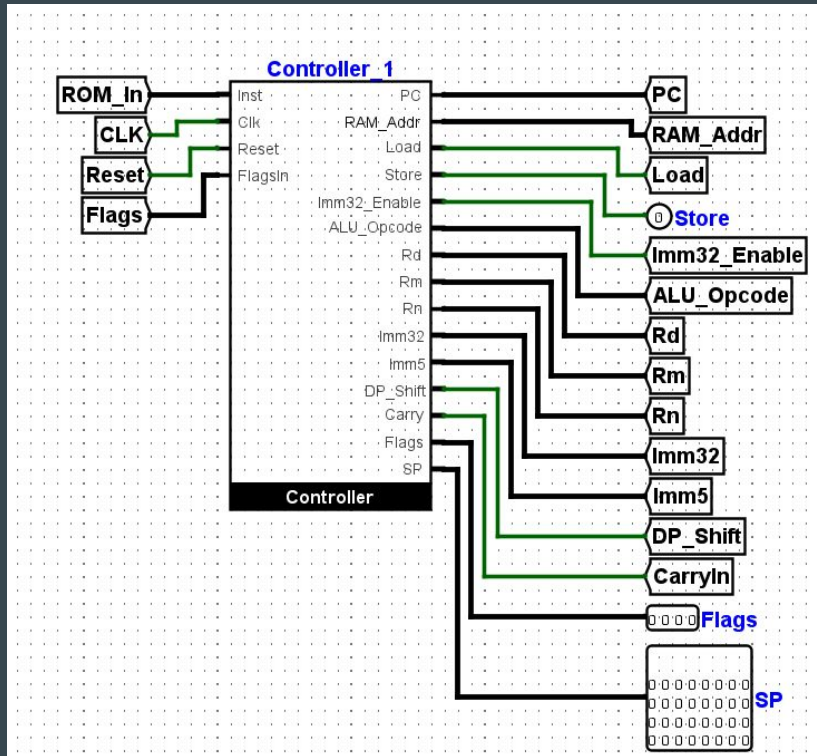
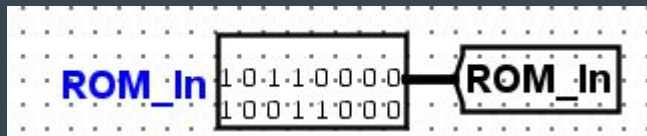
Les composants du processeur:



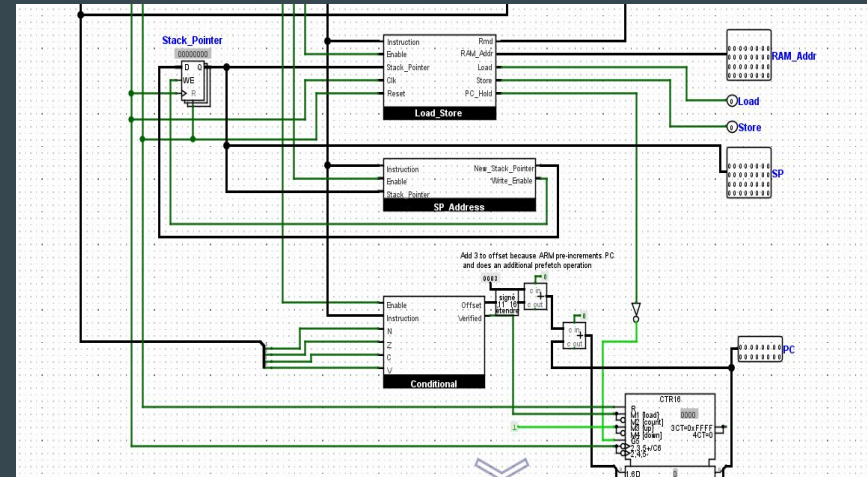
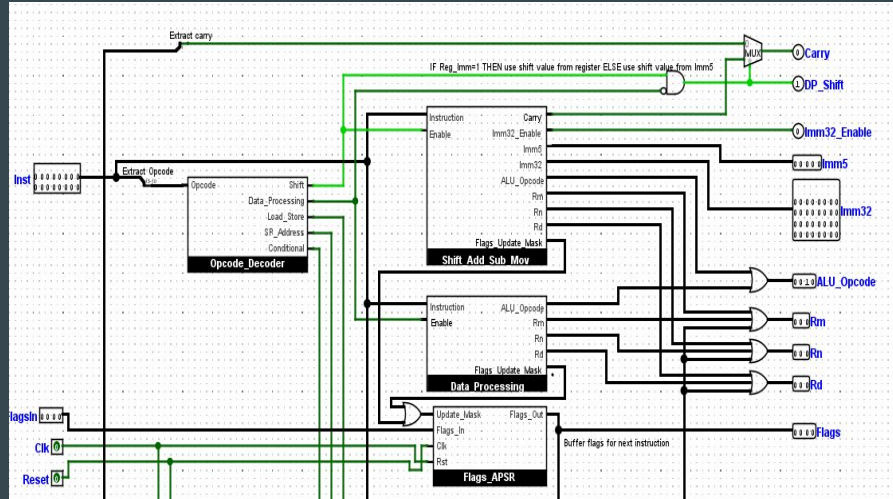
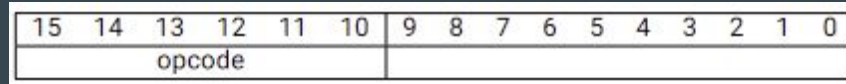
Processeur sur Logisim: La machine Logisim



Processeur sur Logisim: Le contrôleur



Processeur sur Logisim: Le contrôleur



Processeur sur Logisim: Le Opcode Décodeur

9.1.1 Shift, add, sub, mov

Binaire :

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

9.1.2 Data processing

Binaire :

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

9.1.3 Load/Store

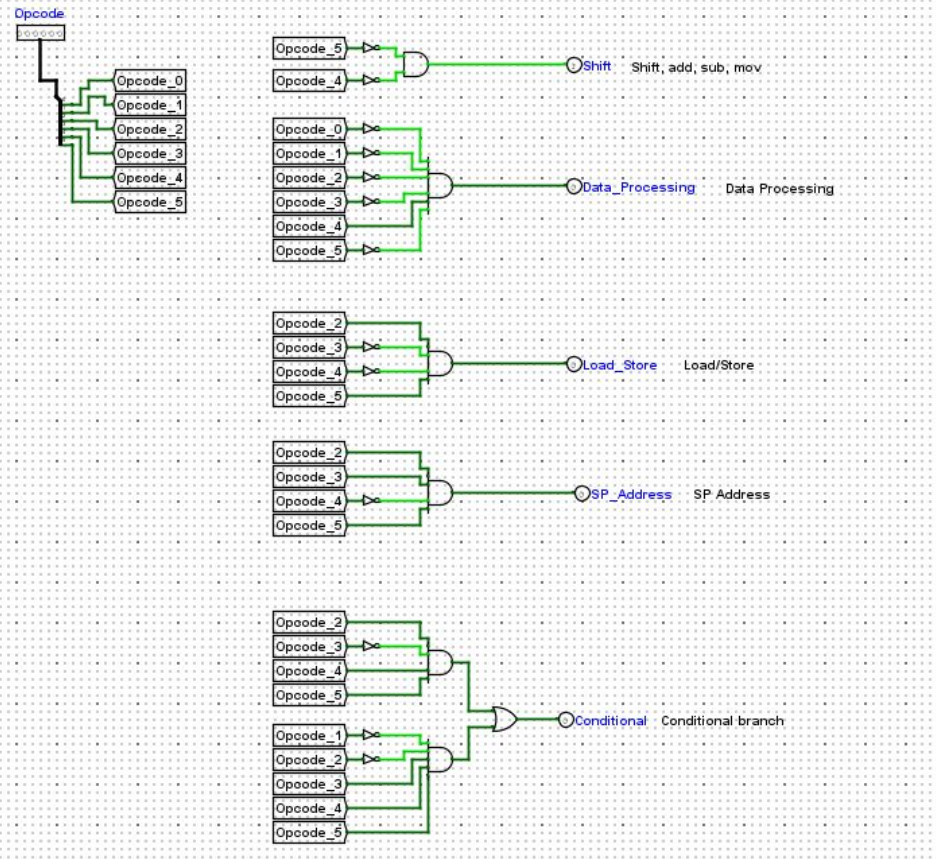
Binaire :

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

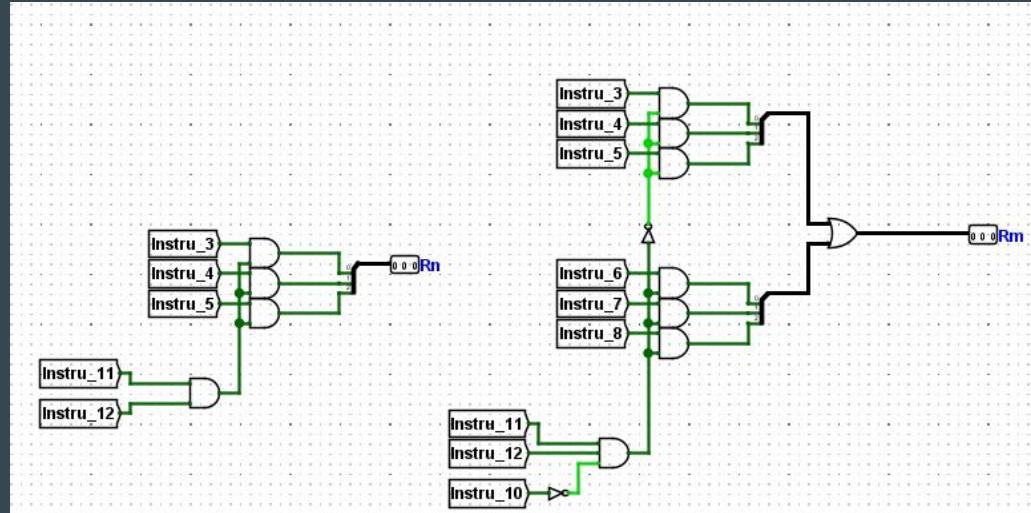
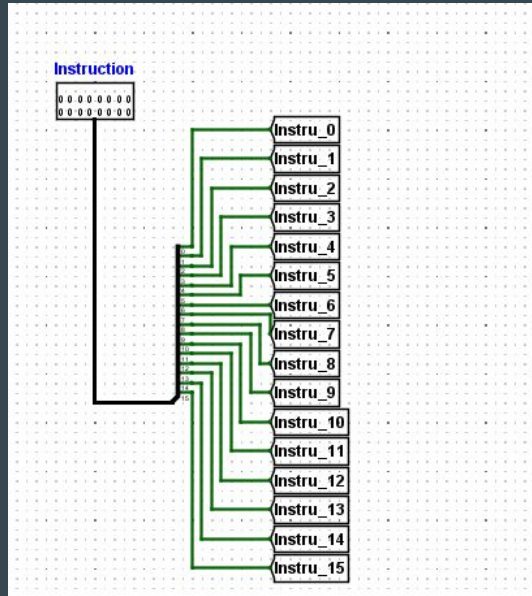
9.1.4 Miscellaneous 16-bit instructions

Binaire :

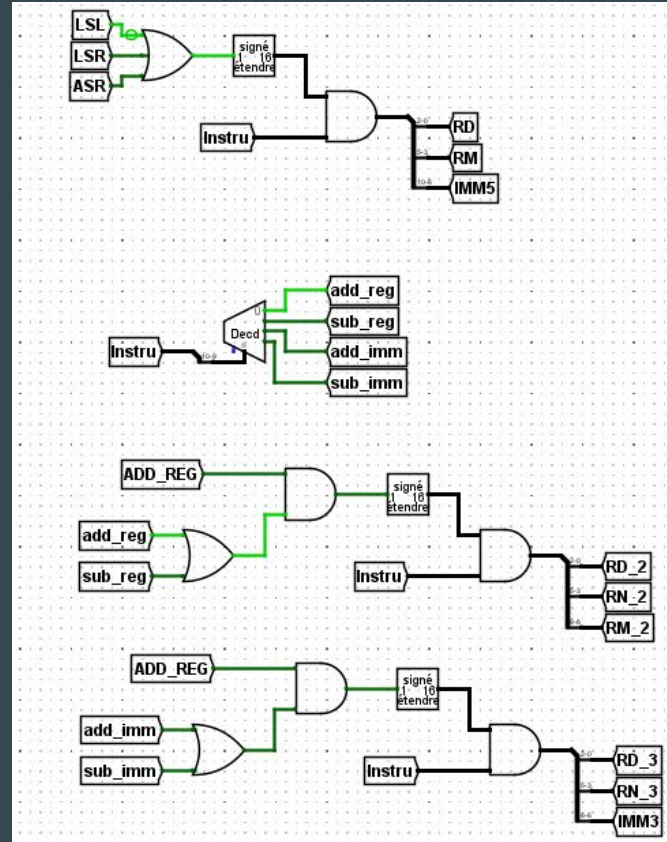
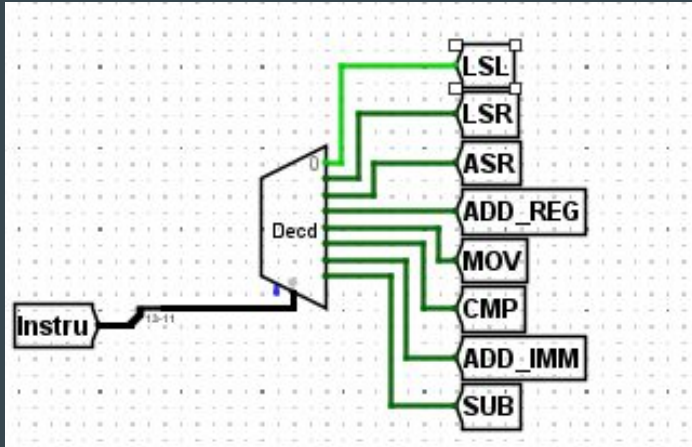
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	opcode											



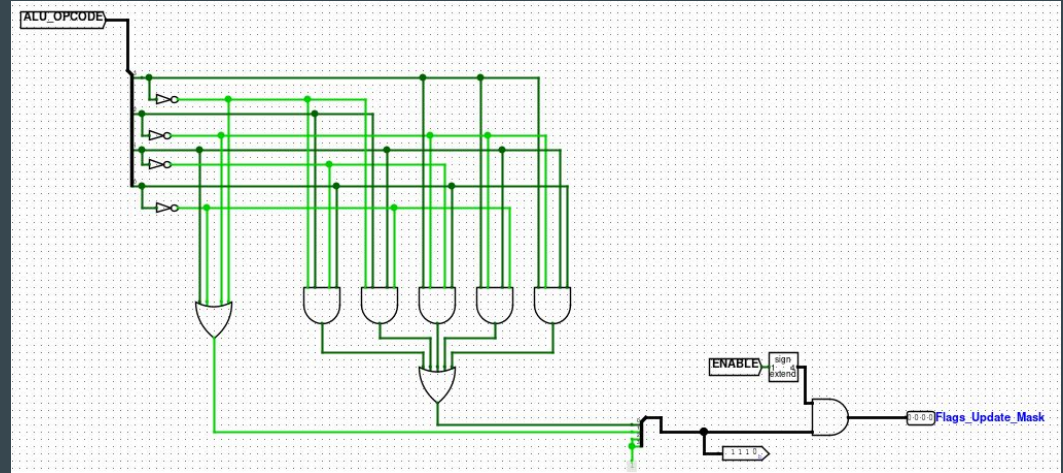
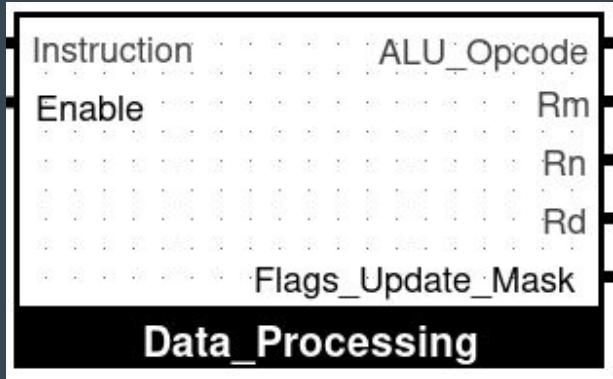
Début de la création des circuits : shift add sub move



Amélioration du circuit : shift add sub move

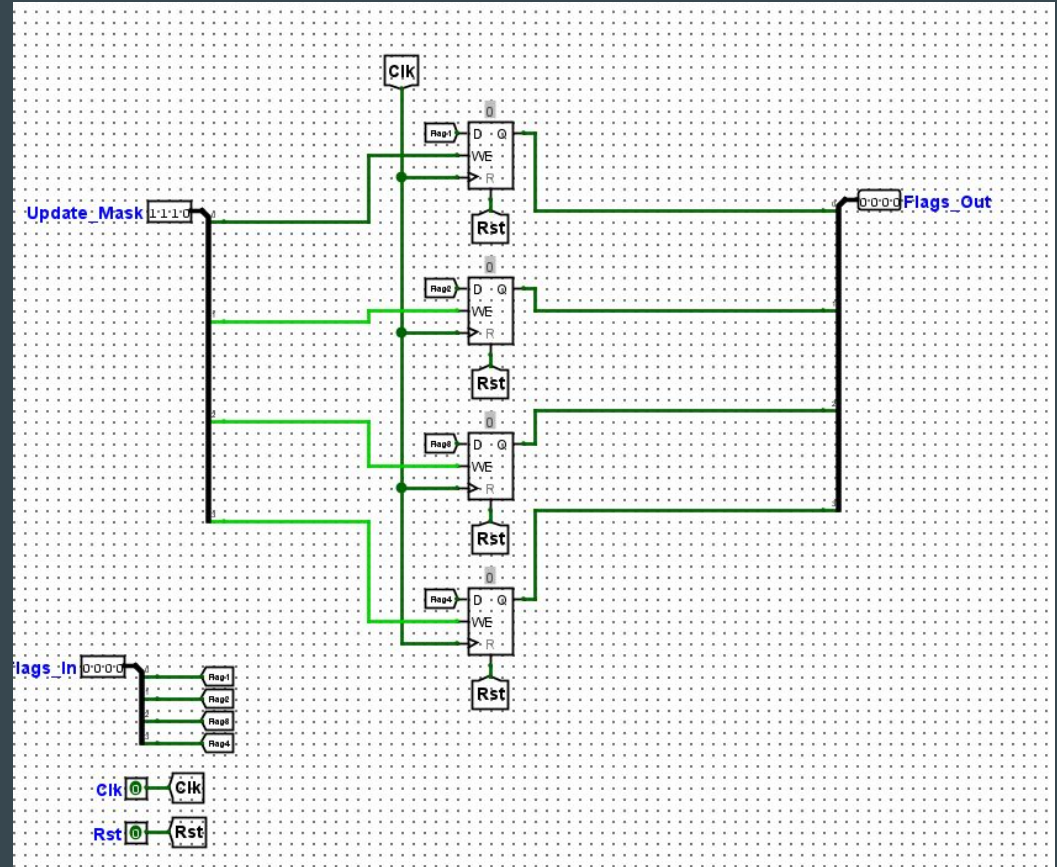


Data processing

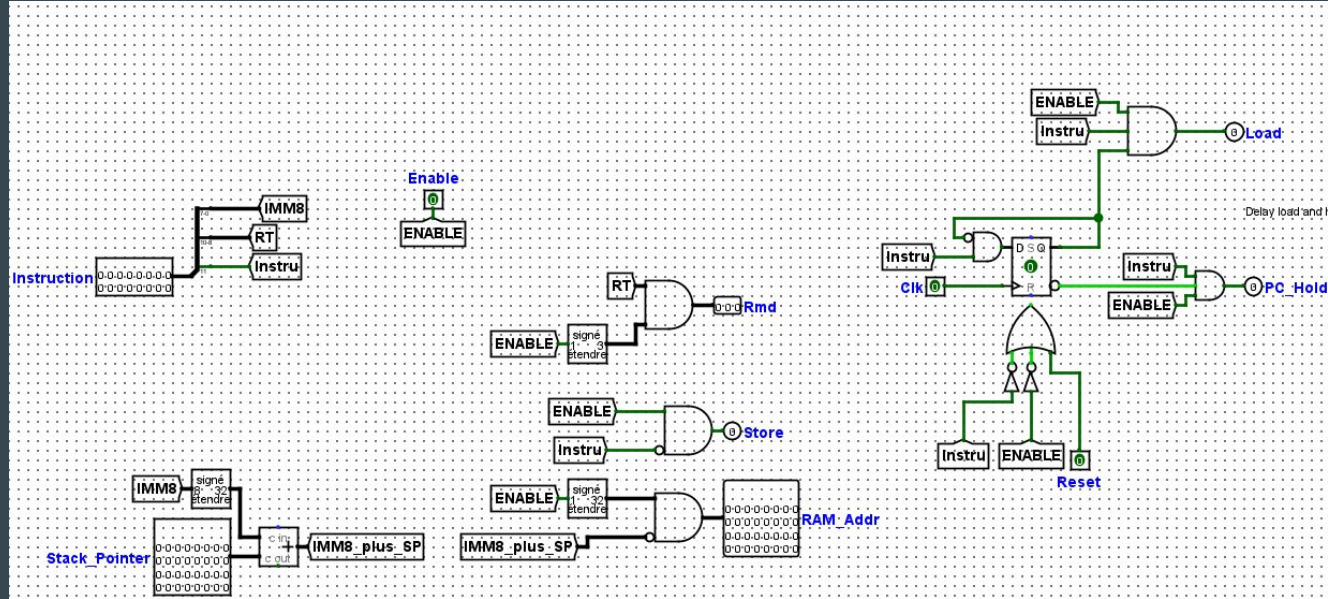
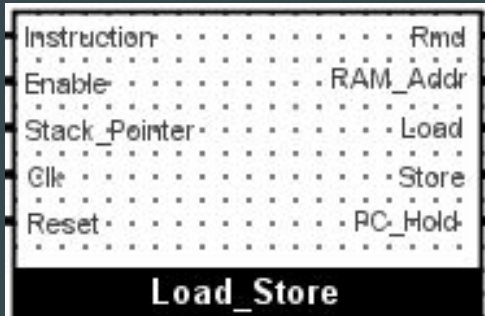


15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	opcode									

Flags_ASPR

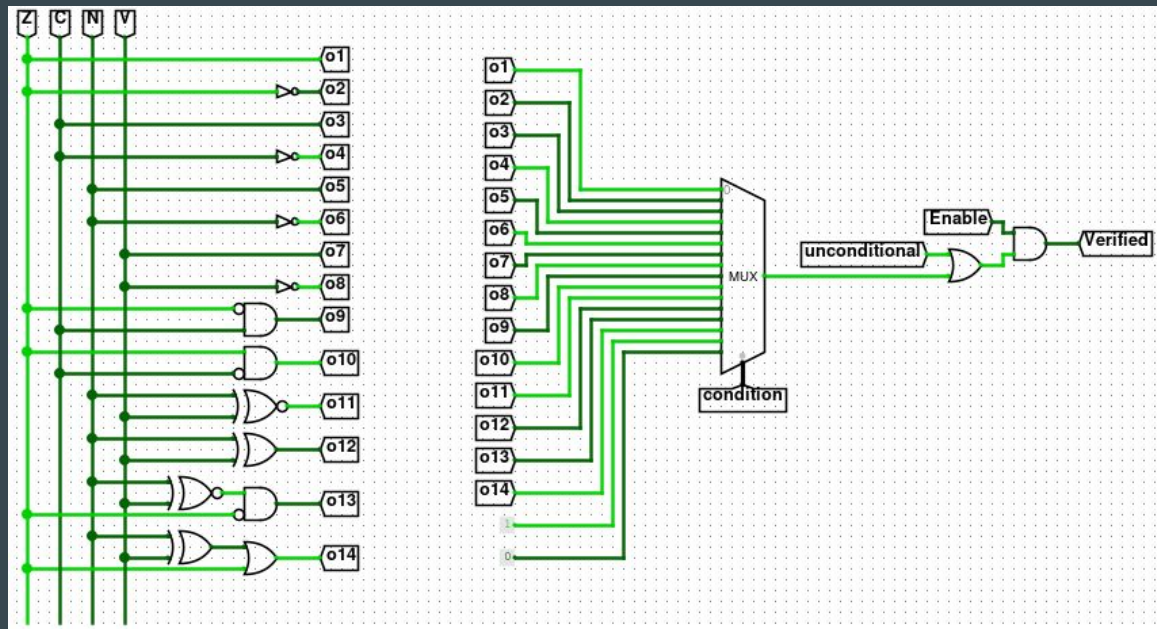


Load Store



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

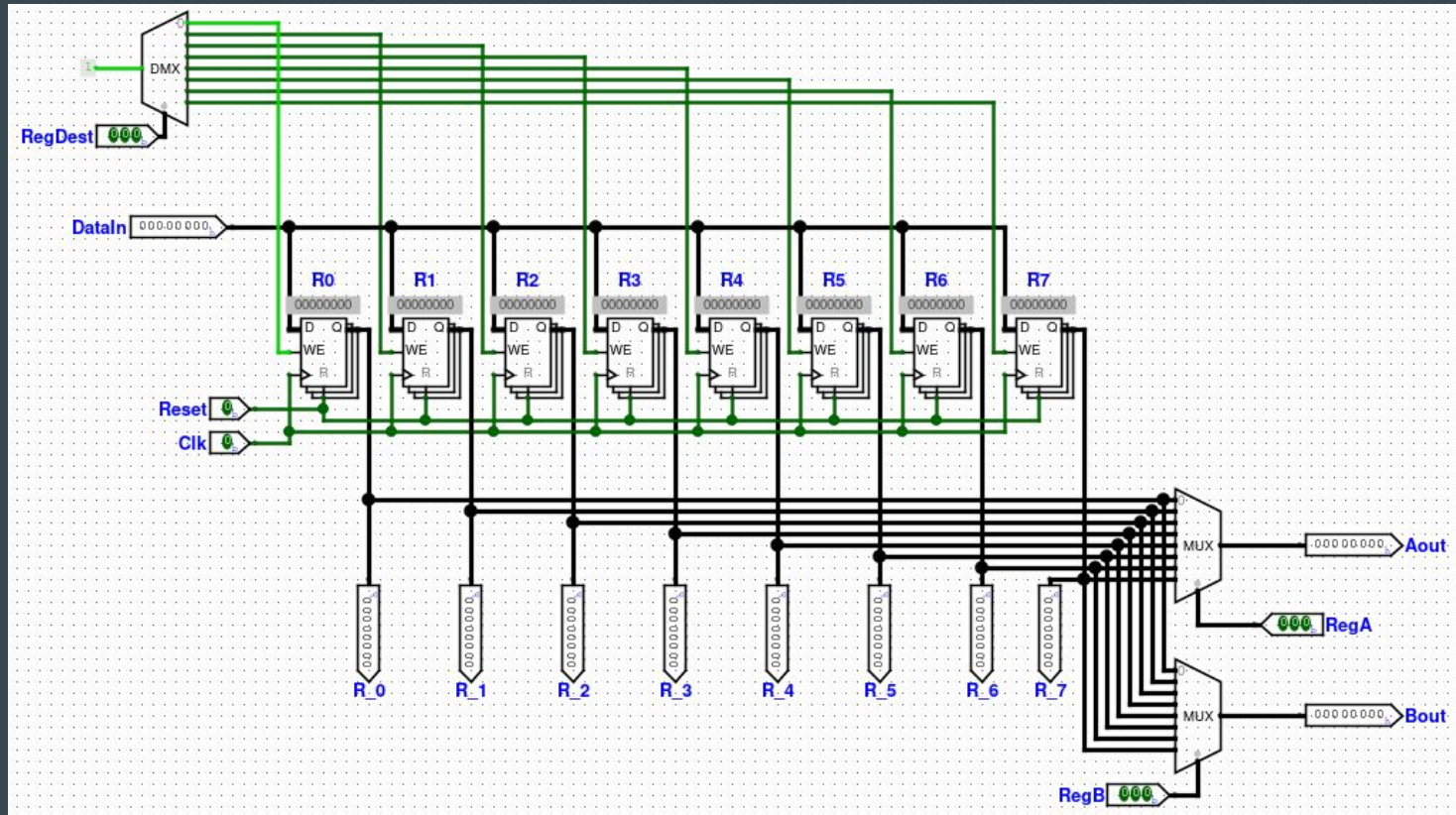
Enable	Offset
Instruction	Verified
N	
Z	
C	
V	
Conditional	



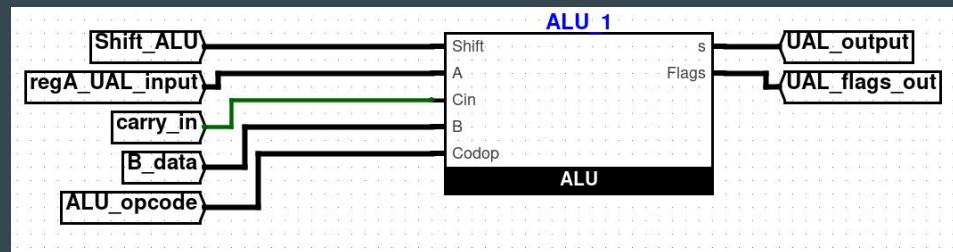
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	imm11										

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	cond				imm8							

Banc de registre



Unité Arithmétique et Logique



Codop	Opération	Instructions	Remarque
0000	A and B	AND	
0001	A xor B	EOR	
0010	B << Shift	LSL	Retenue sortante, voir jeu d'instructions
0011	B >> Shift	LSR	Retenue sortante, voir jeu d'instructions
0100	B >> Shift (arith)	ASR	Retenue sortante, voir jeu d'instructions
0101	A + B + CarryIn	ADC	
0110	B - A + CarryIn - 1	SBC	Retenue entrante inversée
0111	B >> Shift (rot)	ROR	Retenue sortante, voir jeu d'instructions
1000	A and B	TST	Résultat perdu, seuls les drapeaux sont mis à jour
1001	-A	RSB	Registre Rm utilisé plutôt que Rn
1010	B - A	CMP	Résultat perdu, seuls les drapeaux sont mis à jour
1011	A + B	CMN	Résultat perdu, seuls les drapeaux sont mis à jour
1100	A or B	ORR	
1101	A * B	MUL	
1110	B and not A	BIC	
1111	not A	MVN	Complément binaire

Tests

Pour conclure : quelques points à améliorer

- Améliorer la propreté du code du Compilateur
- + Utiliser les propriétés POO de Java pour séparer les tâches
- Réalisation d'un script Bash permettant directement de passer du code C en fichier binaire
- Fixer les bugs spécifiques sur le processeur