

# Projet : Microprocesseur

Joseph Amigo, Alexandre Duplessis, François Ollivier

24 janvier 2021

## Part I ISA

(L'ISA est inspirée de l'architecture RISC-V.)

### 1 Set d'opérations (assembleur)

26 opérations implémentées.

#### 1.1 Opérations arithmétiques et logiques

Keyword	Nom	Op_code	Détail	IC	Rq
add	Addition	0000000000	$rd = ra + rb$	Rdba	
sub	Soustraction	0000000001	$rd = ra - rb$	Rdba	
prod	Produit	0000000010	$rd = ra * rb$	Rdba	NI
div	Division	0000000011	$rd = ra // rb$	Rdba	NI
xor	XOR	0000000100	$rd = ra \text{ XOR } rb$	Rdba	
or	OR	0000000101	$rd = ra \text{ OR } rb$	Rdba	
and	AND	0000000110	$rd = ra \text{ AND } rb$	Rdba	
sll	Shift Left	0000001000	$rd = ra \ll rb$	Rdba	
srl	Shift Right	0000001001	$rd = ra \gg rb$	Rdba	
addi	Addition imm	0000010000	$rd = ra + imm$	Rdai	
subi	Soustraction imm	0000010001	$rd = ra - imm$	Rdai	
xori	XOR imm	0000010100	$rd = ra \text{ XOR } imm$	Rdai	
ori	OR imm	0000010101	$rd = ra \text{ OR } imm$	Rdai	
andi	AND imm	0000010110	$rd = ra \text{ AND } imm$	Rdai	
slli	Shift Left imm	0000011000	$rd = ra \ll imm$	Rdai	
srli	Shift Right imm	0000011001	$rd = ra \gg imm$	Rdai	
not	NOT	0000000111	$rd = \text{NOT } ra$	Rda	

\*IC = Instruction Code

\*NI = Non Implémenté

## 1.2 Opérations sur la mémoire

Keyword	Nom	Op_code	Détail	IC	Rq
load	Load	0001000000	rd = RAM[imm]	Rdi	NI
limm	Load imm	0000110000	rd = imm	Rdi	
store	Store	0001000001	RAM[imm] = ra	Rai	NI
move	Move	0000100000	rd = rb	Rdb	

## 1.3 Opérations de branchement

Keyword	Nom	Op_code	Détail	IC	Rq
beq	Branch ==	0100000001	if (ra == rb) : PC = imm	Rbai	
bne	Branch !=	0110000001	if (ra ≠ rb) : PC = imm	Rbai	
blt	Branch <	1000000001	if (ra > rb) : PC = imm	Rbai	
bge	Branch ≤	1010000001	if (ra ≥ rb) : PC = imm	Rbai	
jump	Jump	0010000000	PC = imm	Ri	

\*PC = Program Counter

## 2 Set d'instructions

Les instructions sont codées sur 38 bits.

37	28	27	24	23	20	19	16	15	0	Code	Exemple
op_code	rd	rb	ra	NU						Rdba	Add
op_code	NU	rb	ra	imm						Rbai	Branch
op_code	rd	NU	NU	imm						Rdi	Load imm
op_code	NU	NU	ra	imm						Rai	Store
op_code	rd	rb	NU	NU						Rdb	Move
op_code	rd	NU	ra	imm						Rdai	Add imm
op_code	NU	NU	NU	imm						Ri	Jump

\*NU = Non Utilisé

\*imm = immediate value

## 3 Architecture

### 3.1 Registres

Choix : 16 registres de 16 bits + 1 registre réservé au PC.

→ adresse se code sur 4 bits

### 3.2 RAM

On n'utilise qu'une seule RAM, et les entiers sont codés sur 16 bits. On n'accepte qu'un accès par cycle.

### 3.3 ROM

La ROM stocke les instructions. L'utilisateur n'y a pas accès (on a déjà un accès par cycle par le processeur). Le PC donne l'adresse de l'instruction à exécuter.

## Part II

# Implémentation

## 1 Processeur

Le processeur en Python. On utilise le compilateur Carotte au lieu de Minijazz.

## 2 Simulateur de netlist

On a écrit un compilateur netlist vers C++ pour des questions d'optimisation.

Le simulateur gère toutes les opérations du langage netlist, le nombre de RAM désiré, une ROM, et des registres.

On procède en deux passes : une pour simuler toutes les équations sauf l'écriture dans la RAM, et une deuxième pour écrire dans la RAM et mettre à jour les registres. En effet ceux-ci sont gérés grâce à deux variables pour chaque variable à stocker, une pour le cycle en cours et une pour le cycle précédent.

Toutes les variables sont stockées sous forme de bitsets.

La ROM est lue dans un fichier *rom* sous forme d'une ligne par mots, les lignes étant séparées par des retours à la ligne.

## 3 Horloge

Elle gère les secondes, les minutes, les heures, les jours, les mois et les années. (On a fait attention au nombre de jours par mois, ainsi qu'aux années bissextiles.)

## 4 Interface graphique

Elle est codée en python avec qt.

## References

[Référence RISC-V] The RISC-V Instruction Set Manual, Andrew Waterman, Krste Asanovic  
(<https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf>)

[Sujet du projet] <https://github.com/hbens/sysnum-2021>