8 registres : rax, rbx, rcx, rdx, rex, rfx, rhx, rz (registre 0).
registres 16 bits.

rz	000
rax	001
rbx	010
rcx	011
rdx	100
rex	101
rfx	110
rhx	111

Instructions sur 32 bits.
On réduit la taille de RISC-V:

- `opcode` sur 3 bits
- `funct3` sur 3 bits
- `funct7` sur 1 bit
- `rd`, `rs2`, `rs1` sur 3 bits (8 registres)
- `imm` sur 16 bits

Arithmétique signée seulement : ADD, SUB, XOR, OR, AND, NAND, Shift (left logical, right logical, right arith).
Load, store 32b.
Branch, jump.

`imm`	`funct7`	`funct3`	`rs2`	`rs1`	`rd`	`opcode`

Inst	Nom	Opcode	funct3	funct7	Description
add	ADD	011	000	0	`rd = rs1 + rs2`
sub	SUB	011	000	1	`rd = rs1 - rs2`
or	OR	011	100	0	`rd = rs1 or rs2`
xor	XOR	011	110	0	`rd = rs1 xor rs2`
and	AND	011	101	0	`rd = (rs1 and rs2)`
nand	NAND	011	100	1	`rd = rs1 nand rs2`
nor	NOR	011	101	1	`rd = rs1 nor rs2`
nxor	NXOR	011	110	1	`rd = rs1 nxor rs2`
sll	Shift left logical	011	011	0	`rd = rs1 « rs2`
srl	Shift right logical	011	010	0	`rd = rs1 » rs2`
sra	Shift right Arith	011	010	1	`rd = rs1 » rs2`
slt	Set less than	011	001	1	`rd = (rs1 < rs2)?1:0`
addi	ADD	001	000	0	`rd = rs1 + imm`
subi	SUB	001	000	1	`rd = rs1 - imm`
ori	OR	001	100	0	`rd = rs1 or imm`
xori	XOR	001	110	0	`rd = rs1 xor imm`
andi	AND	001	101	0	`rd = rs1 and imm`
nandi	NAND	001	100	1	`rd = rs1 nand imm`
nori	NOR	001	101	1	`rd = rs1 nor imm`
nxori	NXOR	001	110	1	`rd = rs1 nxor imm`
slli	Shift left logical	001	011	0	`rd = rs1 « imm`
srli	Shift right logical	001	010	0	`rd = rs1 » imm`
srai	Shift right Arith	001	010	1	`rd = rs1 » imm`
slti	Set less than	001	001	1	`rd = (rs1 < imm)?1:0`
lw	Load word	000	000	0	`rd=M[rs1]`
sw	Store word	010	000	0	`M[rs1]=rs2`
beq	Branch ==	110	000	0	`if(rs1 == rs2) PC=imm`
bne	Branch !=	110	000	1	`if(rs1 != rs2) PC=imm`
ble	Branch ⩽	110	001	0	`if(rs1 <= rs2) PC=imm`
blt	Branch <	110	001	1	`if(rs1 < rs2) PC=imm`
bge	Branch ⩾	110	011	0	`if(rs1 >= rs2) PC=imm`
bgt	Branch >	110	011	1	`if(rs1 > rs2) PC=imm`
jal	Jump and link	101	000	0	`rd=PC; PC=imm`
jalr	Jump and link reg	111	000	0	`rd=PC; PC=rs1+imm`