Le décodage des instructions se fait en 2 étages pour simplifier la modélisation (limité la taille des ROMs), en effet Les ids correspondent au codes qui sont renvoyés pa r le premier étage de décodage. Le code est l'entrée de la ROM de décodage des instructions. Il correspond à la concaténation de opcode[6..2]+funct7[5]+funct3[2..0]. Une seule valeur de code est donnée dans le tableau, mais dans la plupart des cas, `funct7` n'est pas utilisé et donc plusieurs codes sont possibles (bit 4 non contraint).

id	nom	fonction	fmt	opcode	func3	funct7	code	description
1	add	ADD	R	0110011	0x0	0x00	C0	rd = rs1 + rs2
2	sub	SUB	R	0110011	0x0	0x20	C8	rd = rs1 - rs2
3	xor	XOR	R	0110011	0x4	0x00	C4	rd = rs1 ^ rs2
4	or	OR	R	0110011	0x6	0x00	C6	rd = rs1 | rs2
5	and	AND	R	0110011	0x7	0x00	C7	rd = rs1 & rs2
6	sll	Shift Left Logical	R	0110011	0x1	0x00	C1	rd = rs1 << rs2
7	srl	Shift Right Logical	R	0110011	0x5	0x00	C5	rd = rs1 >> rs2
8	sra	Shift Right Arith*	R	0110011	0x5	0x20	CD	rd = rs1 >> rs2
9	slt	Set Less Than	R	0110011	0x2	0x00	C2	rd = (rs1 < rs2)?1:0
A	sltu	Set Less Than (U)	R	0110011	0x3	0x00	C3	rd = (rs1 < rs2)?1:0
B	addi	ADD Immediate	I	0010011	0x0		40	rd = rs1 + imm
C	xori	XOR Immediate	I	0010011	0x4		44	rd = rs1 ^ imm
D	ori	OR Immediate	I	0010011	0x6		46	rd = rs1 | imm
E	andi	AND Immediate	I	0010011	0x7		47	rd = rs1 & imm
F	slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	41	rd = rs1 << imm[0:4]
10	srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	45	rd = rs1 >> imm[0:4]
11	srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	4D	rd = rs1 >> imm[0:4]
12	slti	Set Less Than Imm	I	0010011	0x2		42	rd = (rs1 < imm)?1:0

id	nom	fonction	fmt	opcode	func3	funct7	code	description
13	sltiu	Set Less Than Imm (U)	I	0010011	0x3		43	rd = (rs1 < imm)?1:0
14	lb	Load Byte	I	0000011	0x0		00	rd = M[rs1+imm][0:7]
15	lh	Load Half	I	0000011	0x1		01	rd = M[rs1+imm][0:15]
16	lw	Load Word	I	0000011	0x2		02	rd = M[rs1+imm][0:31]
17	lbu	Load Byte (U)	I	0000011	0x4		04	rd = M[rs1+imm][0:7]
18	lhu	Load Half (U)	I	0000011	0x5		05	rd = M[rs1+imm][0:15]
19	sb	Store Byte	S	0100011	0x0		80	M[rs1+imm][0:7] = rs2[0:7]
1A	sh	Store Half	S	0100011	0x1		81	M[rs1+imm][0:15] = rs2[0:15]
1B	sw	Store Word	S	0100011	0x2		82	M[rs1+imm][0:31] = rs2[0:31]
1C	beq	Branch ==	B	1100011	0x0		180	if(rs1 == rs2) PC += imm
1D	bne	Branch !=	B	1100011	0x1		181	if(rs1 != rs2) PC += imm
1E	blt	Branch <	B	1100011	0x4		184	if(rs1 < rs2) PC += imm
1F	bge	Branch >=	B	1100011	0x5		185	if(rs1 >= rs2) PC += imm
20	bltu	Branch < (U)	B	1100011	0x6		186	if(rs1 < rs2) PC += imm
21	bgeu	Branch >=(U)	B	1100011	0x7		187	if(rs1 >= rs2) PC += imm
22	jal	Jump And Link	J	1101111			1Bx	rd = PC+4; PC += imm
23	jalr	Jump And Link Reg	I	1100111	0x0		190	rd = PC+4; PC = rs1 + imm

id	nom	fonction	fmt	opcode	func3	funct7	code	description
24	lui	Load Upper Imm	U	`0110111`			`0Dx`	`rd = imm << 12`
25	auipc	Add Upper Imm to PC	U	`0010111`			`05x`	`rd = PC + (imm << 12)`
26	ecall/wfi	Environment Call	I	`1110011`	`0x0`	`imm=0x0`	`1C0`	`Transfer control to OS`
26	ebreak	Environment Break	I	`1110011`	`0x0`	`imm=0x1`	`1C0`	`Transfer control to debugger`

Les 2 instr uctions `ecall` et `ebreak` partagent le même id, car il n'est pas possible de les distinguer uniquement avec les bits de l'opcode/funct3 et le bit 6 de funct7. Ce n'est pas un problème, d'autant plus que ces instructions ne seront pas modélisées dans logisim.