

# Decoding RISC-V instructions

Louis Geoffroy

January 2022

## 1 Introduction

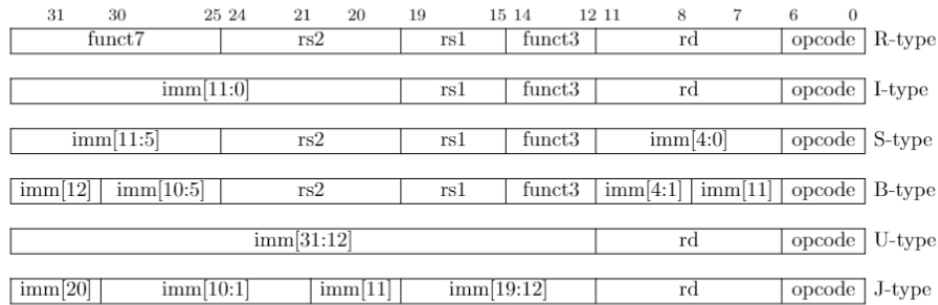


Figure 1: RISC-V instructions de base sans extension

### 1.0.1 R-type :

Les instructions de type R sont des instructions arithmétiques qui utilisent uniquement des registres :

31	25	24	20	19	15	14	12	11	7	6	0
funct7			rs2		rs1	funct3		rd		opcode	
7	5		5		3		5		7		
0000000	src2		src1		ADD/SLT/SLTU		dest		OP		
0000000	src2		src1		AND/OR/XOR		dest		OP		
0000000	src2		src1		SLL/SRL		dest		OP		
0100000	src2		src1		SUB/SRA		dest		OP		

Figure 2: RISC-V instructions de type R

- funct7 : donne des informations sur l'instruction effectuée
- rs2 : indique le registre source numéro 2
- rs1 : indique le registre source numéro 1
- funct3 : indique le typé d'opération que l'on doit effectuer
- rd : registre destination
- opcode = 0110011 : indique que l'on est dans des opérations de type I

Détail des opérations :

- ADD :  $rd = rs1 + rs2$ , overflow ignoré
  - SUB :  $rd = rs2 - rs1$ , overflow ignoré
  - SLT :  $rd = 1$  si  $rs1 < rs2$ , 0 sinon; la comparaison est signée
  - SLTU : comme SLT mais la comparaison n'est pas signée
- Warning : SLTU rd, r0, rs2 effectue l'opération suivante :  $rd = 1$  si  $rs2 \neq 0$  sinon 0

- AND, OR, XOR : opération logique associée - SLL : shift logic left sur rs1 par la valeur contenue dans les 5 bits de poids faible de rs2
- SRL : shift logic right sur rs1 par la valeur contenue dans les 5 bits de poids faible de rs2
- SRA : shift arithmetic right sur rs1 par la valeur contenue dans les 5 bits de poids faible de rs2

Instruction	funct7	funct3	opcode
ADD	0000000	000	0110011
SLT	0000000	010	0110011
SLTU	0000000	011	0110011
AND	0000000	111	0110011
OR	0000000	110	0110011
XOR	0000000	100	0110011
SLL	0000000	001	0110011
SRL	0000000	101	0110011
SUB	0100000	000	0110011
SRA	0100000	101	0110011

### 1.0.2 I-type :

Ce sont des instructions où l'opérande 2 est de type immédiat, on considère dans ce cas un immédiat de 12 bits.

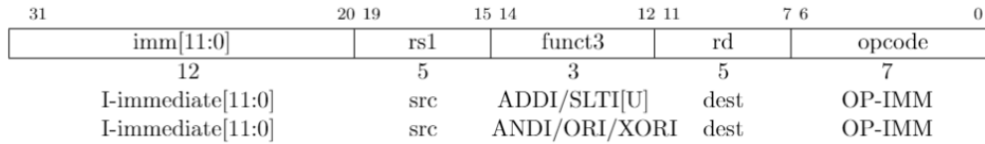


Figure 3: RISC-V instructions de type I

- imm[11:0] : immédiat sur 12 bits que l'on va charger dans l'opérande 2
- rs1 : registre source
- funct3 : code le type d'opération effectuée
- rd : registre destination
- opcode : indique que l'on est dans des opérations de type I

Détail des opérations :

- ADDI : ajoute l'immédiat de 12 bits signed extend à rs1, overflow ignoré
- SLTI (set less than immediate) : rd = 1 si rs1 < (signed extend imm[11:0]) sinon 0
- SLTIU : idem que SLTI mais dans ce cas comparaison est unsigned, l'immédiat est d'abord sign-extend mais est traité comme un entier unsigned
- ANDI, ORI, XORI : opération logique qui font l'opération rd = rs1 op\_logique immediat (sign extend)

Instruction	funct3	opcode
ADDI	000	0010011
SLTI	010	0010011
SLTIU	011	0010011
ANDI	111	0010011
ORI	110	0010011
XORI	100	0010011
JALR	000	1100111

On trouve en plus dans les instructions de type I l'opération NOP qui est définie comme suit :

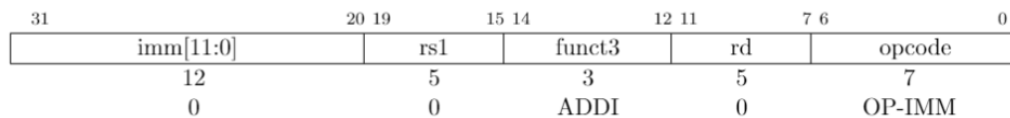


Figure 4: RISC-V NOP inscution de type I

NOP est encodé comme ADDI r0,r0,0

On trouve également l'instruction JALR qui est de type I :

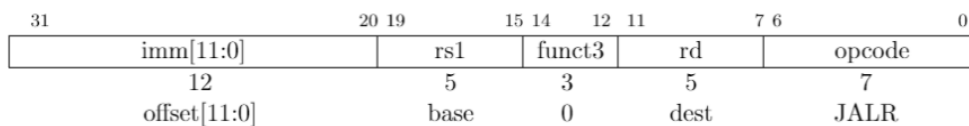


Figure 5: RISC-V inscution jalr de type I

Dans le cas de l'instruction JALR, l'immédiat est signé.

- rd = PC +4, adresse suivant le branchement.
- PC = rs1 + immédiat signé étendu sur 32 bits. Le dernier bit du résultat est mis à 0 pour des raisons d'alignement mémoire.

### 1.0.3 S-type :

Ce sont des instructions de type shift.

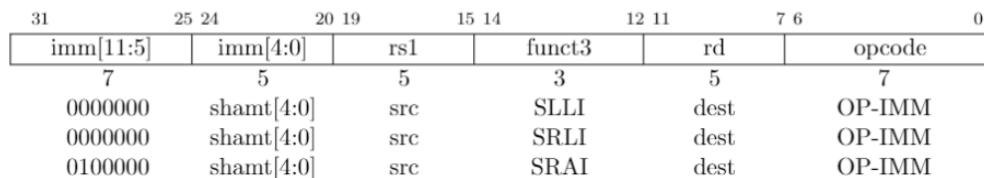


Figure 6: RISC-V inscutions de type S

- imm[11:5] : ? apparemment ne servent à rien
- imm[4:0] : valeur du shift

- rs1 : opérande sur laquelle on va appliquer le shift
- funct3 : type de shift
- rd : registre destination
- opcode : indique que l'on est dans des opérations de type I

Détail des instructions :

- SLLI : Shift logique gauche
- SRLI : Shift logique droite
- SRAI : Shift arithmétique droite

Instruction	funct3	opcode
SLLI	001	0010011
SRLI	101	0010011
SRAI	101	0010011

#### 1.0.4 B-type :

Les instructions de type B sont les instructions de branchement conditionnel :

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12 10:5]		src2	src1	BEQ/BNE	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BLT[U]	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BGE[U]	offset[11 4:1]		BRANCH		

Figure 7: RISC-V instructions de type B

Dans chacune des instructions de branchement conditionnelles une comparaison entre le registres rs1 et rs2 est effectuée.

Détail des instructions :

- BEQ :  $PC = PC + OFFSET$  si  $rs1 = rs2$
- BNE :  $PC = PC + OFFSET$  si  $rs1 \neq rs2$
- BLT :  $PC = PC + OFFSET$  si  $rs1 < rs2$
- BLTU : idem que BLT mais BLTU utilise une comparaison non signée.
- BGE :  $PC = PC + OFFSET$  si  $rs1 \geq rs2$
- BGEU : idem avec comparaison non signée

BGT/BGTU, BLE/BLEU peuvent être obtenu en inversant les opérandes de BLT/BLTU et BGE/BGEU.

Instruction	funct3	opcode
BEQ	000	1100011
BNE	001	1100011
BLT	100	1100011
BGE	101	1100011
BLTU	110	1100011
BGEU	111	1100011

### 1.0.5 U-type :

Les instructions de types U sont les suivantes, elles permettent de charger une valeur dans les bits de poids fort d'un registre. AUIPC a l'air assez particulier :

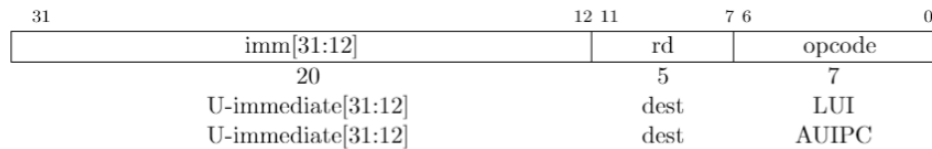


Figure 8: RISC-V instructions de type U

- imm[31:12] : immédiat sur 20 bits
- rd : registre destination
- opcode : indique que l'on est dans des opérations de type I

Détail des instructions :

- LUI (load upper immediate) : charge les 20 bits de l'immédiat dans les 20 bits de poids fort de rd,
- AUIPC (add upper immediate to PC) :  $rd = PC + imm[31:12] \ll 12$ , permet de générer une adresse relative.

Instruction	opcode
LUI	0110111
AUIPC	0110111

### 1.0.6 J-type :

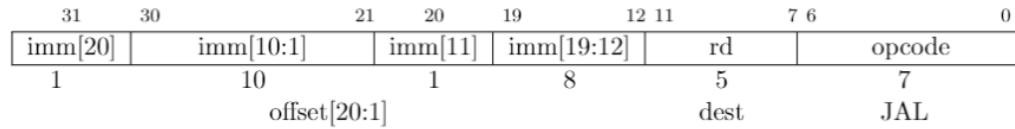


Figure 9: RISC-V inscriptions de type J

- imm[20] imm[10:1] imm[11] imm[19:12] : immédiat signé codant l'offset
- rd : registre desination
- opcode : indique que l'on est dans des opérations de type J

Détail des instructions :

- JAL (jump and link) : l'immédiat encode un offset signé décalé de 2 octets. L'offset est ensuite ajouté à l'adresse contenue dans PC. L'instruction suivant le jump (PC+4) est sauvegardé dans le registre rd.

Détail des prédictions de branchement et de comment jal influe sur RAS page 22-23 de la documentation RISC-V.

Instruction	opcode
JAL	1101111

### 1.0.7 Acces mémoire :

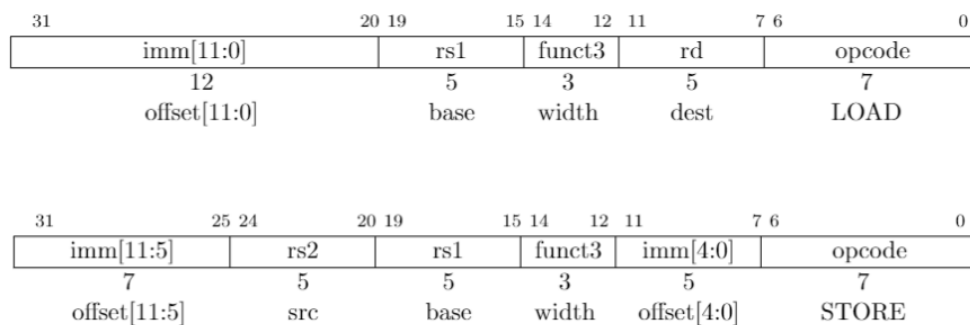


Figure 10: RISC-V inscriptions de type accès mémoire

Les instructions load sont encodés avec le format d'instruction I et les store sont encodés avec le format S.

Détail des instructions :

- LW : load un entier 32 bits depuis la mémoire dans rd,
- LH : load 16 bits depuis la mémoire en puis le signe étend sur 32 bits et le stocke dans rd,
- LHU : load 16 bits mais ne signe étends pas, il zéro étend, puis le stocke dans rd,
- LB : load 8 bits et signe étends, - LBU : load 8 bits et zéro étends,
- SW : idem que LW mais store,
- SH : idem que LH mais store,
- SB : idem que LB mais store,

Instruction	funct3	opcode
LW	010	0000011
LH	001	0000011
LHU	101	0000011
LB	000	0000011
LBU	100	0000011
SW	010	0100011
SH	001	0100011
SB	000	0100011

## 1.1 Conclusion :

Instruction type
R-Type 0110011
I-Type 0010011
S-Type 0010011
B-Type 1100011
U-Type 0110111
J-Type 1101111
Attention pour les accès mémoire malgré que l'encodage se face comme pour I et S, les opcodes s