Автор: Васильев В.И.
Место: Россия, г. С.-Петербург

Дата созд.: 14.11.2025
Дата ред.: 25.11.2025
Версия: 0.04

Ссылки:
1. https://t.me/setun_1958 //[Электронный ресурс] .- группа «Ternary Computers: The Setun and the Setun 70» , тема «SOFT Лабораторные работы»
2. https://www.trinary.su //[Электронный ресурс] .- Троичная логика и троичная цифровая техника.

Таблица 2 : TRIT-RISC-V (32-trits) instruction formats

31:25	24:20	19:15	14:12	11:7	6:0	
func7	rs2	rs1	**func3**	rd	op	R-TYPE
imm11:0		rs1	**func3**	rd	op	I-TYPE
imm11:5	rs2	rs1	**func3**	imm4:0	op	S-TYPE
imm12,10:5	rs2	rs1	**func3**	imm4:1,11	op	SB-TYPE
imm31:12				rd	op	U-TYPE
imm20,10:1,11,19:12				rd	op	UJ-TYPE
fs3 func2	fs2	fs1	**func3**	fd	op	R4-TYPE
5trits 2trits	5trits	5trits	3trits	5trits	7trits	32 trits

Instruction formats details

* R-TYPE Register-register ALU instructions : tadd, txor, tmul
* I-TYPE Immediate ALU instructions, load instructions :taddi,tlw, tjalr, tslli
* S-TYPE Store instructions : tsw, tsb
* SB-TYPE Comparison and branch instructions: tbeq, tbge
* U-TYPE Instructions with upper immediates
* UJ-TYPES Jump instructions: tjal

* **func2** type of operation on 2trits
* **func3** type of operation on 3trits
* **func4** type of operation on 4trits
* **func6** type of operation on 6trits
* **func7** type of operation on 7trits

Glossary of instruction descriptions

* **rs1, rs2** Register descriptors* : Source operands 1 and 2
* **rd** Register descriptor* : Destination operand
* **op** Operation code

* Register descriptors (rs1,rs2, rd) always need 5 trits to address working registers (from t0 to t31).
* That is partly why it is not possible to use CSR directly into regular instructions.
* In “Priviledge / CSR instructions” that are all I-TYPE, the csr operand is coded by imm11:0 (on 12 trits), and that is what theorically allows to address up to 12 trits CSR.
* In addition, in the “Priviledge / CSR instructions”, the 5-trit unsigned immediate (uimm) is coded in the rs1 field and not in the imm11:0 field as it should be because of its previous use.

* **imm**	signed immediate in imm11:0
* **uimm**	5-trits unsigned immediate in imm4:0
* **upimm**	20 upper trirs of a 32-trit immediate, in imm31:12
* **Address**	memory address : rs1 + SignExt(imm11:0)
* **[Address]**	data at memory location Address
* **BTA**	branch target address : PC + SignExt({imm12:1, 1’t0})
* **JTA**	jump target address : PC + SignExt({imm20:1, 1’t0})
* **Label**	text indicating instruction address
* **SignExt**	value sign-extended to 32 trits
* **ZeroExt**	value zero-extended to 32 trits
* **csr**	constrol and status register

Краткое описание набора команд TRIT-RISC-V

Таблица 3 : TRIT-RV32I RISC-V Integer instructions

op	Func3	Func7	Type	Mnemonic		Description	Operation
0000011(3), **t00000++**(4)	**000**(0),**t000**(0)		**I**	**tLB**	rd, imm(rs1)	**Load** tryte	**rd** = **SignExt**([Address]7:0)
0000011(3), **t00000++**(4)	**001**(1),**t00+**(0)		**I**	**tLH**	rd, imm(rs1)	**Load** half	**rd** = **SignExt**([Address]15:0)
0000011(3), **t00000++**(4)	**010**(2),**t0+0**(3)		**I**	**tLW**	rd, imm(rs1)	**Load** word	**rd** = ([Address]31:0)
0000011(3), **t00000++**(4)	**100**(4),**t+00**(9)		**I**	**tLBU**	rd, imm(rs1)	**Load** byte unsigned	**rd** = **ZeroExt**([Address]7:0)
0000011(3), **t00000++**(4)	**101**(5),**t+0+**(10)		**I**	**tLHU**	rd, imm(rs1)	**Load** half unsigned	**rd** = **ZeroExt**([Address]15:0)
0010011(19), **t00+00++**(85)	**000**(0),**t000**(0)		**I**	**tADDI**	rd, rs1, imm	**ADD** immediate	**rd** = **rs1** + **SignExt**(imm)
0010011(19), **t00+00++**(85)	**001**(1),**t00+**(1)		**I**	**tSLLI**	rd, rs1, uimm	**Shift left** logical immediate	**rd** = **rs1** << uimm
0010011(19), **t00+00++**(85)	**010**(2),**t0+0**(3)		**I**	**tSLTI**	rd, rs1, imm	**Set** less than immediate	**rd** = **rs1** < SignExt(imm)
0010011(19), **t00+00++**(85)	**011**(3),**t0++**(4)	**0000000**(0),**t0000000**(0)	**I**	**tSLTIU**	rd, rs1, imm	**Set** less than imm. unsigned	**rd** = **rs1** < SignExt(imm)
0010011(19), **t00+00++**(85)	**100**(4),**t+00**(9)		**I**	**tXORI**	rd, rs1, imm	**XOR** immediate	**rd** = **rs1** ^ SignExt(imm)
0010011(19), **t00+00++**(85)	**101**(4),**t+0+**(10)	**0000000**(0),**t0000000**(0)	**I**	**tSRLI**	rd, rs1, imm	**Shift right** logical immediate	rd = rs1 >> uimm
0010011(19), **t00+00++**(85)	**101**(4),**t+0+**(10)	**0100000**(32),**t0+00000**(243)	**I**	**tSRAI**	rd, rs1, imm	**Shift right** arithmetic immediate	rd = rs1 >> uimm
0010011(19), **t00+00++**(85)	**110**(6),**t++0**(12)		**I**	**tORI**	rd, rs1, imm	**OR** immediate	**rd** = **rs1** | SignExt(imm)
0010011(19), **t00+00++**(85)	**111**(7),**t+++**(13)		**I**	**tANDI**	rd, rs1, imm	**AND** immediate	**rd** = **rs1** & SignExt(imm)
0010111(23), **t00+0+++**(94)			**U**	**tAUIPC**	rd, rs1, imm	**ADD** upper immediate to PC	**rd** = (**upimm**, 12’t0) + **PC**
0100011(35), **t0+000++**(247)			**S**	**tSB**	rs2,imm(rs1)	Store byte	**[Address]**7:0 = **rs2**7:0
0100011(35), **t0+000++**(247)			**S**	**tSH**	rs2,imm(rs1)	Store half	**[Address]**15:0 = **rs**215:0
0100011(35), **t0+000++**(247)			**S**	**tSW**	rs2,imm(rs1)	Store word	**[Address]**31:0 = **rs**20
0110011(51), **t0++00++**(328)	**000**(0),**t000**(0)	**0000000**(0),**t0000000**(0)	**R**	**tADD**	rd, rs1, rs2	**tADD**	**rd** = **rs1** + **rs2**

0110011(51), **t0++00++**(328)	**000**(0),**t000**(0)	**0100000**(32),**t0+00000**(243)	**R**	**tSUB**	rd, rs1, rs2	**tSUB**	**rd** = **rs1 - rs2**
0110011(51), **t0++00++**(328)	**001**(1),**t00+**(1)	**0000000**(0),**t0000000**(0)	**R**	**tSLL**	rd, rs1, rs2	**Shift left** logical immediate	**rd** = **rs1** << **rs2**4:0
0110011(51), **t0++00++**(328)	**010**(2),**t0+0**(3)	**0000000**(0),**t0000000**(0)	**R**	**tSLT**	rd, rs1, rs2	**Set less** than	**rd** = **rs1** < **rs2**
0110011(51), **t0++00++**(328)	**011**(3),**t0++**(4)	**0000000**(0),**t0000000**(0)	**R**	**tSLTU**	rd, rs1, rs2	**Set less** than unsigned	**rd** = **rs1** < **rs2**
0110011(51), **t0++00++**(328)	**100**(4),**t+00**(9)	**0000000**(0),**t0000000**(0)	**R**	**tXOR**	rd, rs1, rs2	**tXOR**	**rd** = **rs1** ^ **rs2**
0110011(51), **t0++00++**(328)	**101**(5),**t+0+**(10)	**0000000**(0),**t0000000**(0)	**R**	**tSRL**	rd, rs1, rs2	**Shift right** logical immediate	**rd** = **rs1** >> **rs2**4:0
0110011(51), **t0++00++**(328)	**101**(5),**t+0+**(10)	**0100000**(32),**t0+00000**(243)	**R**	**tSRAI**	rd, rs1, rs2	**Shift right arithmetic immediate**	**rd** = **rs1** >>> **rs2**4:0
0110011(51), **t0++00++**(328)	**110**(6),**t++0**(12)	**0000000**(0),**t0000000**(0)	**R**	**tOR**	rd, rs1, rs2	**tOR**	**rd** = **rs1** | **rs2**
0110011(51), **t0++00++**(328)	**111**(7),**t+++**(13)	**0000000**(0),**t0000000**(0)	**R**	**tAND**	rd, rs1, rs2	**tAND**	**rd** = **rs1** & **rs2**
0110111(55), **t0++0+++**(337)			**U**	**tLUI**	rd, upimm	Load upper immediate	**rd** = {**upimm**, 12't0}
1100011(99), **t++000++**(976)	**000**(0),**t000**(0)		**B**	**tBEQ**	rs1,rs2, label	**Branch** if equal =	if (rs1 == rs2) PC = BTA
1100011(99), **t++000++**(976)	**001**(0),**t00+**(1)		**B**	**tBNE**	rs1,rs2, label	**Branch** if not equal ≠	if (rs1 != rs2) PC = BTA
1100011(99), **t++000++**(976)	**010**(2),**t0+0**(3)		**B**	**tBLT**	rs1,rs2, label	**Branch** if lower than <	if (rs1 < rs2) PC = BTA
1100011(99), **t++000++**(976)	**011**(3),**t0++**(4)		**B**	**tBGE**	rs1,rs2, label	**Branch** if greater / equal ≥	if (rs1 ≥ rs2) PC = BTA
1100011(99), **t++000++**(976)	**100**(4),**t+00**(9)		**B**	**tBLTU**	rs1,rs2, label	Branch if lower than unsigned <	if (rs1 < rs2) PC = BTA
1100011(99), **t++000++**(976)	**101**(5),**t+0+**(10)		**B**	**tBGEU**	rs1,rs2, label	Branch if greater / equal unsign. ≥	if (rs1 < rs2) PC = BTA
1100111(103), **t++00+++**(985)	**000**(0),**t000**(0)		**I**	**tJALR**	rd, rs1, label	**Jump** and link register	PC = rs1 + SignExt(imm) rd = PC + 4
1101111(111), **t++0++++**(1012)			**J**	**tJAL**	rd, label	**Jump** and link	PC = JTA rd = PC + 4

Таблица 4 : Common TRIT-RV32I RISC-V Integer pseudoinstructions

Pseudoinstruction	RISC-V Instructions	Description	Operation
nop	taddi x0, x0, 0	no operation	
tli rd,imm11:0	taddi rd, x0, imm11:0	load 12-bit immediate	rd =SignExtend(imm11:0)
tlir d,imm31:0	tlui rd, imm31:12 taddi rd, rd, imm11:0	load 32-bit immediate	rd =imm31:0
tmv rd,rs1	taddi rd, rs1, 0	move (also called “register copy”)	rd =rs1
not rd,rs1	txori rd,rs1, -1	one’s complement	rd = ~rs1
tneg rd,rs1	tsub rd,x0, rs1	two’s complement	rd = —rs1
tseqz rd,rs1	tsltiu rd,rs1, 1	set if = 0	rd = (rs1 == 0)
tsnez rd,rs1	tsltu rd, x0, rs1	set if ≠ 0	rd = (rs1 ≠0)
tsltz rd,rs1	tslt rd,rs1, x0	set if < 0	rd = (rs1 <0)
tsgtz rd,rs1	tslt rd,x0,rs1	set if > 0	rd = (rs1 >0)

tbeqz rs1, label	tbeqr s1, x0, label	branch if = 0	if (rs1 == 0)
			PC = label

tbnez rs1, label	tbner s1, x0,label	branch if ≠ 0	if (rs1 ≠0)PC = label
tblez rs1, label	tbge x0, rs1, label	branch if ≤ 0	if (rs1 ≤0)PC = label
tbgez rs1, label	tbge rs1, x0,label	branch if ≥ 0	if (rs1 ≥0)PC = label
tbltz rs1, label	tblt rs1, x0,label	branch if < 0	if (rs1 <0)PC = label
tbgtz rs1, label	tblt x0, rs1, label	branch if > 0	if (rs1 >0)PC = label
tbleu rs1, rs2, label	tbgeu rs2, rs1, label	branch if ≤ (unsigned)	if (rs1 ≤rs2) PC = label
tbgtu rs1, rs2, label	tbltu rs2, rs1, offset	branch if > (unsigned)	if (rs1 >rs2) PC = label
tbgtu rs1, rs2, label	tbltu rs2, rs1, offset	branch if > (unsigned)	if (rs1 >rs2) PC = label
tj label	tjal x0,label	jump	PC = label

tjal label	rjal ra,label	jump and link	PC = label,
			ra = PC + 4

tjrr s1	tjalr x0,rs1, 0	jump register	PC = rs1

tjalr rs1	tjalr ra,rs1, 0	jump and link register	PC = rs1, ra = PC + 4
ret	tjalr x0,rs1, 0	return from function	PC = ra
call label	tjal ra,label	call nearby function	PC = label, ra = PC + 4
call label	tauipc ra, tjalr ra,offset31:12 Tra, offset11:0	call far away function	PC = PC + offset, ra = PC + 4
la rd,symbol	tauipc ra, jalr ra,offset31:12 ra, offset11:0	load address of global variable	rd = PC + symbol
l{b|h|w} rd,symbol	tauipc rd, symbol	load address of global variable	rd = PC + symbol
s{b|h|w} rs2, symbol, rs1	tauipc rs1, symbol31:12 ts{b|h|w} rs2, symbol11	store global variable	[PC + symbol] = rs2
tcsrr rd, csr	tcsrrs rd, csr, x0	read CSR	rd = csr
tcsrw csr, rs1	tcsrrw x0, csr, rs1	write CSR	csr = rs1
tcsrs/tcsrc csr, rs1	tcsrrs/tcsrrc x0, csr, rs1	set/clear bits in CSR	csr = csr | rs1 / csr = csr & ~rs1

* if trit 11 of the immediate/offset/symbol is 1, the upper immediate is incremented by 1. offset/symbol are the 32-trit PC-relative addresses of a label/global variable.