RISC-V RV64I Simple Green Card

Instruction	Type	Opcode	Funct3	Funct7/IMM	Operation
add rd, rs1, rs2	R	0x33	0x0	0x00	R[rd] ← R[rs1] + R[rs2]
mul rd, rs1, rs2			0x0	0x01	R[rd] ← (R[rs1] * R[rs2])[63:0]
sub rd, rs1, rs2			0x0	0x20	R[rd] ← R[rs1] - R[rs2]
sll rd, rs1, rs2			0x1	0x00	R[rd] ← R[rs1] << R[rs2
mulh rd, rs1, rs2			0x1	0x01	R[rd] ← (R[rs1] * R[rs2])[127:64]
slt rd, rs1, rs2			0x2	0x00	R[rd] ← (R[rs1] < R[rs2]) ? 1 : 0
xor rd, rs1, rs2			0x4	0x00	R[rd] ← R[rs1] ^ R[rs2]
div rd, rs1, rs2			0x4	0x01	R[rd] ← R[rs1] / R[rs2]
srl rd, rs1, rs2			0x5	0x00	R[rd] ← R[rs1] >> R[rs2]
sra rd, rs1, rs2			0x5	0x20	R[rd] ← R[rs1] >> R[rs2]
or rd, rs1, rs2			0x6	0x00	R[rd] ← R[rs1] | R[rs2]
rem rd, rs1, rs2			0x6	0x01	R[rd] ← (R[rs1] % R[rs2]
and rd, rs1, rs2			0x7	0x00	R[rd] ← R[rs1] & R[rs2]
lb rd, offset(rs1)	I	0x03	0x0		R[rd] ← SignExt(Mem(R[rs1] + offset, byte))
lh rd, offset(rs1)			0x1		R[rd] ← SignExt(Mem(R[rs1] + offset, half))
lw rd, offset(rs1)			0x2		R[rd] ← SignExt(Mem(R[rs1] + offset, word))
ld rd, offset(rs1)			ox3		R[rd] ← Mem(R[rs1] + offset, doubleword)
addi rd, rs1, imm		0x13	0x0		R[rd] ← R[rs1] + imm
slli rd, rs1, imm			0x1	0x00	R[rd] ← R[rs1] << imm[5:0]
slti rd, rs1, imm			0x2		R[rd] ← (R[rs1] < imm) ? 1 : 0
xori rd, rs1, imm			0x4		R[rd] ← R[rs1] ^ imm
srli rd, rs1, imm			0x5	0x00	R[rd] ← R[rs1] >> imm[5:0]
srai rd, rs1, imm			0x5	0x10	R[rd] ← R[rs1] >> imm[5:0]
ori rd, rs1, imm			0x6		R[rd] ← R[rs1] | imm
andi rd, rs1, imm			0x7		R[rd] ← R[rs1] & imm
addiw rd, rs1, imm		0x1B	0x0		R[rd] ← SignExt((R[rs1](63:0) +SignExt(imm))[31:0])
Jalr rd, rs1, imm		0x67	0x0		R[rd] ← PC + 4
					PC ← { (R[rs1] + imm), 1b'0}
ecall		0x73	0x0	0x000	(Transfers control to operating system)
					a0 = 1 is print value of a1 as an integer.
					a0 = 10 is exit or end of code indicator.
sb rs2, offset(rs1)	S	0x23	0x0		Mem(R[rs1] + offset) ← R[rs2][7:0]
sh rs2, offset(rs1)			0x1		Mem(R[rs1] + offset) ← R[rs2][15:0]
sw rs2, offset(rs1)			0x2		Mem(R[rs1] + offset) ← R[rs2][31:0]
sd rs2, offset(rs1)			0x3		Mem(R[rs1] + offset) ← R[rs2][63:0]
beq rs1, rs2, offset	SB	0x63	0x0		if(R[rs1] == R[rs2])
					PC ← PC + {offset, 1b'0}
bne rs1, rs2, offset			0x1		if(R[rs1] != R[rs2])
					PC ← PC + {offset, 1b'0}
blt rs1, rs2, offset			0x4		if(R[rs1] < R[rs2])
					PC ← PC + {offset, 1b'0}
bge rs1, rs2, offset			0x5		if(R[rs1] >= R[rs2])
					PC ← PC + {offset, 1b'0}
auipc rd, offset	U	0x17			R[rd] ← PC + {offset, 12'b0}
lui rd, offset		0x37			R[rd] ← {offset, 12'b0}
jal rd, imm	UJ	0x6f			R[rd] ← PC + 4
					PC ← PC + {imm, 1b'0}

For further reference, here are the bit lengths of the instruction components

R-TYPE	funct7	rs2	rs1	funct3	rd	opcode
Bits	7	5	5	3	5	7

I-TYPE	imm[11:0]	rs1	funct3	rd	opcode
Bits	12	5	3	5	7

S-TYPE	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
Bits	7	5	5	3	5	7

SB-TYPE	imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
Bits	1	6	5	5	3	4	1	7

U-TYPE	imm[31:12]	rd	opcode
Bits	20	5	7

UJ-TYPE	imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
Bits	1	10	1	8	5	7

执行结果参考：
https://kvakil.github.io/venus/

勘误：

v1.1	修改了**addiw**和**lw**的错误
v1.2	修改了mul和mulh的错误
v1.3	SLLI，SRLI,SRAI在rv64下,shamt位数增加为6位
v1.4	与jal不同，jalr是结果最低位补0