RISC-V RV64G Simple Green Card

add rd, rs1, rs2 mul rd, rs1, rs2 mul rd, rs1, rs2 sib rd, rs1, rs2 sib rd, rs1, rs2 sib rd, rs1, rs2 sil rd, rs1, rs2 sil rd, rs1, rs2 sil rd, rs1, rs2 strd, rs2, rs2 str	Instruction	Туре	Opcode		Funct7/IMM		
Dec Dec		Турс	Орсоис				
Sub rd, rs1, rs2 Sil rd, rs2, rs2 Sil rd, r			0x33				
Sil rd, rs1, rs2							
Multh rd, rs1, rs2							
Sit rd, rs1, rs2							
XOT rd, rs1, rs2 Abri rd, rs2 Abri							
div rd, rs1, rs2 sfr rd, rs1, rs2 sfr rd, rs1, rs2 sfr rd, rs1, rs2 or rd, rs1, rs2 or rd, rs1, rs2 or rd, rs1, rs2 or rd, rs1, rs2 ox6 ox00 R[rd] + R[rs1] > R[rs2] ox6 ox00 R[rd] + R[rs1] > R[rs2] ox6 ox00 R[rd] + R[rs1] + R[rs2] ox6 ox00 R[rd] - R[rs1] + R[rs2] ox6 ox01 R[rd] - R[rs1] + R[rs2] ox01 R[rd] - R[rs1] + R[rs2] ox02 R[rd] - R[rs1] + R[rs2] ox03 R[rd] - R[rs1] + R[rs2] ox03 R[rd] - R[rs1] ramm ox1							
SrI rd, rs1, rs2 Sra rd, rs1, rs2 Ox5 Ox00 R[rd] + R[rs1] >> R[rs2]							
Sra rd, rs1, rs2							
or rd, rs1, rs2 ox6 0x00 R[rd] — R[rs1] R[rs2] orem rd, rs1, rs2 ox6 0x01 R[rd] — R[rs1] R[rs2] addw rd, rs1, rs2 ox6 0x01 R[rd] — R[rs1] R[rs2] subw rd, rs1, rs2 ox0 0x00 R[rd] — SignExtt(R[rs1][31:0] + R[rs2][31:0])) mulw rd, rs1, rs2 ox0 ox0 ox0 R[rd] — SignExtt(R[rs1][31:0] + R[rs2][31:0])[31:0]) ibr d, offset(rs1) ox0 ox0 ox0 R[rd] — SignExtt(R[rs1][31:0] + R[rs2][31:0])[31:0]) ibr d, offset(rs1) ox0 ox0 R[rd] — SignExtt(R[rs1][31:0] + R[rs2][31:0])[31:0]) ibr d, offset(rs1) ox0 ox0 R[rd] — SignExtt(R[rs1][31:0] + R[rs2][31:0])[31:0]) ibr d, offset(rs1) ox0 R[rd] — SignExtt(Mem(R[rs1] + offset, bufl)) ibr d, offset(rs1) ox0 R[rd] — SignExtt(Mem(R[rs1] + offset, word)) ox1 addir d, rs1, imm ox1 addir d, rs1, imm stir rd, rs1, imm ox1 addir d, rs1, imm stir rd, rs1, imm ox1 addir d, rs1, imm ori rd, rs1, imm ox1 addir d, rs1, imm ori rd, rs1, imm ox1 addir d, rs1, imm <t< td=""><td></td><td>R</td><td></td><td></td><td></td></t<>		R					
Rem rd, rs1, rs2							
and rd, rs1, rs2 addw rd, rs1, rs2 addw rd, rs1, rs2 subw rd, rs1, rs2 ox0 ox00 ox20							
Decomposition Decompositio							
Subw rd, rs1, rs2 Dx0							
Mulw rd, rs1, rs2 Dox3							
Divw rd, rs1, rs2 Div rd, rs1, rs2 Div rd, rs1, rs2 Div rd, rs1, rs2 Div rd, offset(rs1) Div rd, rs1, imm Stil rd, rs1, imm Stil rd, rs1, imm Stil rd, rs1, imm Stil rd, rs1, imm Div rd, rs2, imm Div rd, rs2, imm Div rd, rs2, imm Div rd, rs3, imm Div rd, rs4, imm			020				
Decomposition Decompositio			OX3B				
District District							
Ih rd, offset(rs1) Iw rd, offset(rs1) Iw rd, offset(rs1) Iw rd, offset(rs1) Id rd, rs1, imm If rd, rs1, imm I							
Iw rd, offset(rs1) Id rd, offset(rs1) Id rd, offset(rs1) addi rd, rs1, imm slli rd, rs1, imm slli rd, rs1, imm slli rd, rs1, imm srli rd, rs1, imm ori rd, rs1, imm addiw rd, rs1, imm addiw rd, rs1, imm addiw rd, rs1, imm addiw rd, rs1, imm ox7 R[rd] + R[rs1] \times imm[5:0] ox6 R[rd] + R[rs1] \times imm[5:0] ox6 R[rd] + R[rs1] \times imm[5:0] ox6 R[rd] + R[rs1] \times imm ox7 ox7 ox7 ox7 ox7 ox7 ox7 ox7 ox7 o							
Id rd, offset(rs1) addi rd, rs1, imm slii rd, rs1, imm slii rd, rs1, imm slii rd, rs1, imm slti rd, rs1, imm stii rd, rs1, imm srii rd, rs1, imm srii rd, rs1, imm srii rd, rs1, imm srii rd, rs1, imm ori rd, rs1, imm ori rd, rs1, imm addiw rd, rs1, imm addiw rd, rs1, imm ox7 R[rd] + R[rs1] \(\) imm ox8 ox8 R[rd] + R[rs1] \(\) imm ox8 R[rd] + R[rs1] \(\) imm			0x03				
Description							
Sili rd, rs1, imm Sili				_			
slti rd, rs1, imm xori rd, rs1, imm sr1 rd, rs1, imm xori rd, rs1, imm srai rd, rs1, imm xori rd, rs1, imm ori rd, rs1, imm xori rd, rs1, imm andi rd, rs1, imm xori rd, rs1, imm addiw rd, rs1, imm xori rd, rs1, imm Jalr rd, rs1, imm xori rd, rs1, imm ox1B xori rd, rs1, imm ox2B xori rd, rs1, imm ox73 xori rd, rs1, rs1, rs1, rs1, rs1, rs1, rs2, rs1, rs2, rs2, rs2, rs2, rs2, rs2, rs2, rs2							
Note					0x00		
srli rd, rs1, imm ox5 0x00 R[rd] ← R[rs1] >> imm[5:0] ori rd, rs1, imm ox6 0x10 R[rd] ← R[rs1] >> imm[5:0] ox6 R[rd] ← R[rs1] imm andi rd, rs1, imm 0x7 R[rd] ← R[rs1] & imm blt rd, rs1, imm 0x0 R[rd] ← PC + 4 call 0x0 0x00 (Transfers control to operating system) 0x73 0x0 0x00 (Transfers control to operating system) 0x1 0x1 0x1 0x2 0x3 0x1 0x2 0x3 0x4 0x4 0x4 0x2 0x3 0x4							
srli rd, rs1, imm ori rd, rs1, imm ox5 0x00 R[rd] ← R[rs1] >> imm[5:0] ori rd, rs1, imm ox6 R[rd] ← R[rs1] imm ox6 R[rd] ← R[rs1] imm addiw rd, rs1, imm ox7 R[rd] ← R[rs1] & imm ox7 R[rd] ← PC + 4 ox67 pc ← R[rs1] + imm (PC[0] = 0) ox0 (Transfers control to operating system) ecall ox0 ox00 (Transfers control to operating system) ox18 ox0 ox00 (Transfers control to operating system) ox19 ox0 ox00 (Transfers control to operating system) ox19 ox2 ox10							
sraird, rs1, imm ori rd, rs1, imm 0x5 0x10 R[rd] ← R[rs1] >> imm[5:0] ori rd, rs1, imm 0x6 R[rd] ← R[rs1] imm addiw rd, rs1, imm 0x1B 0x0 R[rd] ← SignExt((R[rs1][31:0] + SignExt(imm))) Jair rd, rs1, imm 0x0 R[rd] ← PC + 4 PC ← R[rs1] + imm (PC[0] = 0) ecall 0x0 0x000 (Transfers control to operating system) ecall 0x0 0x000 (Transfers control to operating system) 0x73 a0 = 1 is print value of a1 as an integer. sb rs2, offset(rs1) 0x0 Mem(R[rs1] + offset) ← R[rs2][7:0] ox1 Mem(R[rs1] + offset) ← R[rs2][15:0] 0x1 Mem(R[rs1] + offset) ← R[rs2][31:0] 0x2 Mem(R[rs1] + offset) ← R[rs2][63:0] beq rs1, rs2, offset 0x0 if(R[rs1] = R[rs2]) 0x1 if(R[rs1] = R[rs2]) 0x2 PC ← PC + {offset, 1b'0} 0x3 Mem(R[rs1] + offset) ← R[rs2][63:0] 0x4 if(R[rs1] = R[rs2]) 0x4 if(R[rs1] < R[rs2])							
andi rd, rs1, imm addiw rd, rs1, imm 0x7 R[rd] ← R[rs1] & imm Jalr rd, rs1, imm 0x0 R[rd] ← SignExt((R[rs1][31:0] + SignExt(imm))) ecall 0x0 R[rd] ← PC + 4 ecall 0x0 0x0000 (Transfers control to operating system) 0x73 0x0 0x0000 (Transfers control to operating system) 0x1 0x1 0x2 0x3 0x3 0x4 0x5 0x6					0x10		
addiw rd, rs1, imm Ox1B Ox0 R[rd] ← SignExt((R[rs1][31:0] + SignExt(imm))) Jalr rd, rs1, imm Ox67 Ox0 R[rd] ← PC + 4 ecall PC ← R[rs1] + imm (PC[0] = 0) 0x73 0x0 Ox000 (Transfers control to operating system) a0 = 1 is print value of a1 as an integer. 0x0 Mem(R[rs1] + offset) ← R[rs2][7:0] sb 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] ox3 Mem(R[rs1] + offset) ← R[rs2][63:0] beq rs1, rs2, offset 0x0 if(R[rs1] = R[rs2]) Dox0 if(R[rs1] != R[rs2]) PC ← PC + {offset, 1b'0} Ox1 if(R[rs1] < R[rs2])							
Jair rd, rs1, imm 0x67 0x0 R[rd] ← PC + 4 ecall 0x0 0x0000 (Transfers control to operating system) 0x73 a0 = 1 is print value of a1 as an integer. sb rs2, offset(rs1) a0 = 10 is exit or end of code indicator. sh rs2, offset(rs1) 0x0 Mem(R[rs1] + offset) ← R[rs2][7:0] sw rs2, offset(rs1) 0x1 Mem(R[rs1] + offset) ← R[rs2][15:0] ox2 Mem(R[rs1] + offset) ← R[rs2][63:0] ox3 Mem(R[rs1] + offset) ← R[rs2][63:0] beq rs1, rs2, offset 0x0 if(R[rs1] = R[rs2]) ox4 if(R[rs1] = R[rs2]) ox4 if(R[rs1] < R[rs2])	andi rd, rs1, imm					R[rd] ← R[rs1] & imm	
Ox67 PC ← R[rs1] + imm (PC[0] = 0)	addiw rd, rs1, imm		0x67	0x0		$R[rd] \leftarrow SignExt((R[rs1][31:0] + SignExt(imm)))$	
PC ← R[rs1] + imm (PC[0] = 0) ox0	Jalr rd, rs1, imm			0x0		R[rd] ← PC + 4	
0x73 a0 = 1 is print value of a1 as an integer.						$PC \leftarrow R[rs1] + imm (PC[0] = 0)$	
	ecall			0x0	0x000		
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$						a0 = 1 is print value of a1 as an integer.	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$						a0 = 10 is exit or end of code indicator.	
sw rs2, offset(rs1) $0x2$ $Mem(R[rs1] + offset) \leftarrow R[rs2][31:0]$ sd rs2, offset(rs1) $0x3$ $Mem(R[rs1] + offset) \leftarrow R[rs2][63:0]$ beq rs1, rs2, offset $0x0$ $if(R[rs1] == R[rs2])$ $PC \leftarrow PC + \{offset, 1b'0\}$	sb rs2, offset(rs1)		0x23	0x0		$Mem(R[rs1] + offset) \leftarrow R[rs2][7:0]$	
sw rs2, offset(rs1) $0x2$ $Mem(R[rs1] + offset) \leftarrow R[rs2][31:0]$ sd rs2, offset(rs1) $0x3$ $Mem(R[rs1] + offset) \leftarrow R[rs2][63:0]$ beq rs1, rs2, offset $0x0$ $if(R[rs1] == R[rs2])$ bne rs1, rs2, offset $0x1$ $if(R[rs1] != R[rs2])$ $0x4$ $if(R[rs1] < R[rs2])$ $0x4$ $if(R[rs1] < R[rs2])$ $0x4$	sh rs2, offset(rs1)	S		0x1		$Mem(R[rs1] + offset) \leftarrow R[rs2][15:0]$	
beq rs1, rs2, offset Dx0 if(R[rs1] == R[rs2])	sw rs2, offset(rs1)			0x2		$Mem(R[rs1] + offset) \leftarrow R[rs2][31:0]$	
Description	sd rs2, offset(rs1)			0x3		$Mem(R[rs1] + offset) \leftarrow R[rs2][63:0]$	
bne rs1, rs2, offset SB Ox1 if(R[rs1] != R[rs2]) PC ← PC + {offset, 1b'0} Ox4 if(R[rs1] < R[rs2]) PC ← PC + {offset, 1b'0} Ox5 if(R[rs1] >= R[rs2])	beq rs1, rs2, offset	SB	0x63	0x0		if(R[rs1] == R[rs2])	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$						PC ← PC + {offset, 1b'0}	
blt rs1, rs2, offset	bne rs1, rs2, offset			0x1		if(R[rs1] != R[rs2])	
blt rs1, rs2, offset 0x4 if(R[rs1] < R[rs2])						PC ← PC + {offset, 1b'0}	
$PC \leftarrow PC + \{offset, 1b'0\}$ bge rs1, rs2, offset $0x5 \qquad if(R[rs1] >= R[rs2])$	blt rs1, rs2, offset			0x4			
bge rs1, rs2, offset $0x5 if(R[rs1] >= R[rs2])$							
U 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	bge rs1, rs2, offset			0x5			
$PC \leftarrow PC + \{offset, 1b'0\}$						PC ← PC + {offset, 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	1				
							_	
UJ-TYPE	imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode]	

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是结果最低位置0
- v1.5 增加了addw, subw, mulw, divw, remw的说明

auipc rd, offset	U	0x17	R[rd] ← PC + {offset, 12'b0}	
lui rd, offset	0	0x37	R[rd] ← {offset, 12'b0}	
jal rd, imm	UJ	0x6f	R[rd] ← PC + 4	
			PC ← PC + {imm, 1b'0}	