RV32I Base Instruction Set

(Assembled by jinho.rho)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TYPE	MNEMONIC	NAME	Descript	Note
Funct7 (7)							Regitster Source 2 (5)					Regitster Source 1 (5)					Funct3 (3)			Register Destination (5)					Opcode (7)												
0	0	0	0	0	0	0	rs2					rs1					0	0	0	rd					0	1	1	0	0	1	1	R-TYPE	ADD	ADD	rd = rs1 + rs2	
0	1	0	0	0	0	0	rs2					rs1					0	0	0	rd					0	1	1	0	0	1	1		SUB	SUB	rd = rs1 - rs2	
0	0	0	0	0	0	0	rs2					rs1					0	0	1	rd					0	1	1	0	0	1	1		SLL	Shift Left Logical	rd = rs1 << rs2	
0	0	0	0	0	0	0	rs2					rs1					1	0	1	rd					0	1	1	0	0	1	1		SRL	Shift Right Logical	rd = rs1 >> rs2	
0	1	0	0	0	0	0	rs2					rs1					1	0	1	rd					0	1	1	0	0	1	1		SRA	Shift Right Arith*	rd = rs1 >>> rs2	msb-extends
0	0	0	0	0	0	0	rs2					rs1					0	1	0	rd					0	1	1	0	0	1	1		SLT	Set Less Than	rd = (rs1 < rs2) ? 1:0	
0	0	0	0	0	0	0	rs2					rs1					0	1	1	rd					0	1	1	0	0	1	1		SLTU	Set Less Than (U)	rd = (rs1 < rs2) ? 1:0	zero-extends
0	0	0	0	0	0	0	rs2					rs1					1	0	0	rd					0	1	1	0	0	1	1		XOR	XOR	rd = rs1 ^ rs2	
0	0	0	0	0	0	0	rs2					rs1					1	1	0	rd					0	1	1	0	0	1	1		OR	OR	rd = rs1 | rs2	
0	0	0	0	0	0	0	rs2					rs1					1	1	1	rd					0	1	1	0	0	1	1		AND	AND	rd = rs1 & rs2	

```
add x7, x5, x6  # x7 = x5 + x6
sub x7, x5, x6  # x7 = x5 - x6
sll x7, x5, x6  # x7 = x5 << x6
srl x7, x5, x6  # x7 = x5 >> x6
sra x7, x5, x6  # x7 = x5 >>> x6
slt x7, x5, x6  # x7 = 1 if x5 < x6 else 0
sltu x7, x5, x6  # x7 = 1 if x5 < x6 else 0
xor x7, x5, x6  # x7 = x5 ^ x6
or x7, x5, x6  # x7 = x5 | x6
and x7, x5, x6  # x7 = x5 & x6
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TYPE	MNEMONIC	NAME	Descript	Note
immediate (7)							shift amount (5)					Regitster Source 1 (5)					Funct3 (3)			Register Destination (5)					Opcode (7)												
imm[11:0]												rs1					0	0	0	rd					0	0	0	0	0	1	1	L-TYPE	LB	Load Byte	rd = M[rs1+imm][0:7]	
imm[11:0]												rs1					0	0	1	rd					0	0	0	0	0	1	1		LH	Load Half	rd = M[rs1+imm][0:15]	
imm[11:0]												rs1					0	1	0	rd					0	0	0	0	0	1	1		LW	Load Word	rd = M[rs1+imm][0:31]	
imm[11:0]												rs1					1	0	0	rd					0	0	0	0	0	1	1		LBU	Load Byte (U)	rd = M[rs1+imm][0:7]	zero-extends
imm[11:0]												rs1					1	0	1	rd					0	0	0	0	0	1	1		LHU	Load Half (U)	rd = M[rs1+imm][0:15]	zero-extends
imm[11:0]												rs1					0	0	0	rd					0	0	1	0	0	1	1	I-TYPE	ADDI	ADD Immediate	rd = rs1 + imm	
imm[11:0]												rs1					0	1	0	rd					0	0	1	0	0	1	1		SLTI	Set Less Than Imm	rd = (rs1 < imm) ? 1 : 0	
imm[11:0]												rs1					0	1	1	rd					0	0	1	0	0	1	1		SLTIU	Set Less Than Imm (U)	rd = (rs1 < imm) ? 1 : 0	
imm[11:0]												rs1					1	0	0	rd					0	0	1	0	0	1	1		XORI	XOR Immediate	rd = rs1 ^ imm	
imm[11:0]												rs1					1	1	0	rd					0	0	1	0	0	1	1		ORI	OR Immediate	rd = rs1 | imm	
imm[11:0]												rs1					1	1	1	rd					0	0	1	0	0	1	1		ANDI	AND Immediate	rd = rs1 & imm	
0	0	0	0	0	0	0	shamt					rs1					0	0	1	rd					0	0	1	0	0	1	1		SLLI	Shift Left Logical Imm	rd = rs1 << shamt[0:4]	
0	0	0	0	0	0	0	shamt					rs1					1	0	1	rd					0	0	1	0	0	1	1		SRLI	Shift Right Logical Imm	rd = rs1 >> shamt[0:4]	
0	1	0	0	0	0	0	shamt					rs1					1	0	1	rd					0	0	1	0	0	1	1		SRAI	Shift Right Arith Imm	rd = rs1 >>> shamt[0:4]	

```
lb x5, 0(x10)  # Load byte from memory
lh x5, 0(x10)  # Load halfword from memory
lw x5, 0(x10)  # Load word from memory
lbu x7, 4(x10)
lhu x7, 6(x10)
addi x7, x5, 10  # x7 = x5 + 10
slti x7, x5, 10  # x7 = 1 if x5 < 10 else 0
sltiu x7, x5, 10  # x7 = 1 if x5 < 10 else 0
xori x7, x5, 0xFF  # x7 = x5 ^ 0xFF
ori x7, x5, 0x0F  # x7 = x5 | 0x0F
andi x7, x5, 0xF0  # x7 = x5 & 0xF0
slli x7, x5, 4  # x7 = x5 << 4
srli x7, x5, 4  # x7 = x5 >> 4
srai x7, x5, 4  # x7 = x5 >>> 4
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TYPE	MNEMONIC	NAME	Descript	Note
immediate (7)							Regitster Source 2 (5)					Regitster Source 1 (5)					Funct3 (3)			immediate (5)					Opcode (7)												
imm[11:5]							rs2					rs1					0	0	0	imm[4:0]					0	1	0	0	0	1	1	S-TYPE	SB	Store Byte	M[rs1+imm][0:7] = rs2[0:7]	
imm[11:5]							rs2					rs1					0	0	1	imm[4:0]					0	1	0	0	0	1	1		SH	Store Half	M[rs1+imm][0:15] = rs2[0:15]	
imm[11:5]							rs2					rs1					0	1	0	imm[4:0]					0	1	0	0	0	1	1		SW	Store Word	M[rs1+imm][0:31] = rs2[0:31]	

```
sb x6, 0(x10)  # Store byte to memory
sh x6, 0(x10)  # Store halfword to memory
sw x6, 0(x10)  # Store word to memory
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TYPE	MNEMONIC	NAME	Descript	Note
immediate (7)							Regitster Source 2 (5)					Regitster Source 1 (5)					Funct3 (3)			immediate (5)					Opcode (7)												
imm[12][10:5]							rs2					rs1					0	0	0	imm[4:1][11]					1	1	0	0	0	1	1	B-TYPE	BEQ	Branch ==	if(rs1 == rs2) PC += imm	
imm[12][10:5]							rs2					rs1					0	0	1	imm[4:1][11]					1	1	0	0	0	1	1		BNE	Branch !=	if(rs1 != rs2) PC += imm	
imm[12][10:5]							rs2					rs1					1	0	0	imm[4:1][11]					1	1	0	0	0	1	1		BLT	Branch <	if(rs1 < rs2) PC += imm	
imm[12][10:5]							rs2					rs1					1	0	1	imm[4:1][11]					1	1	0	0	0	1	1		BGE	Branch ≥	if(rs1 >= rs2) PC += imm	
imm[12][10:5]							rs2					rs1					1	1	0	imm[4:1][11]					1	1	0	0	0	1	1		BLTU	Branch < (U)	if(rs1 < rs2) PC += imm	
imm[12][10:5]							rs2					rs1					1	1	1	imm[4:1][11]					1	1	0	0	0	1	1		BGEU	Branch ≥ (U)	if(rs1 >= rs2) PC += imm	

```
beq x5, x6, label  # Jump to label if x5 == x6
bne x5, x6, label  # Jump to label if x5 != x6
blt x5, x6, label  # Jump to label if x5 < x6
bge x5, x6, label  # if x5 >= x6, jump
bltu x5, x6, label  # if x5 < x6, jump (unsigned 비교)
bgeu x5, x6, label  # if x5 >= x6, jump (unsigned 비교)
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TYPE	MNEMONIC	NAME	Descript	Note
immediate (20)																				Register Destination (5)					Opcode (7)												
imm[31:12]																				rd					0	1	1	0	1	1	1	LU-TYPE	LUI	Load Upper Imm	rd = imm << 12	
imm[31:12]																				rd					0	0	1	0	1	1	1	AU-TYPE	AUIPC	Add Upper Imm to PC	rd = PC + (imm << 12)	
imm[20][10:1][11][19:12]																				rd					1	1	0	1	1	1	1	J-TYPE	JAL	Jump And Link	rd = PC + 4; PC+= imm	
imm[11:0]												rs1					0	0	0	rd					1	1	0	0	1	1	1	JL-TYPE	JALR	Jump And Link Reg	rd = PC + 4; PC = rs1 + imm	

```
lui x5, 0x10000  # Load upper immediate
auipc x5, 0x10000  # Upper immediate to PC
jal x1, func  # Jump to func, save return address
jalr x1, x5, 0  # Jump to address in x5
```