RV32I R/I/U Instruction Reference

Note: in some cases, instructions are grouped with types other than their actual encoding for logical clarity. Such cases are specifically noted.

Note: all immediate values should be sign-extended. The sign bit of all immediate values is in the most-significant bit (bit 31) of the instruction.

R-Type

These instructions operate on two registers, storing the result in a third register.

31-25	24-20	19-15	14-12	11-7	6-0
`funct7`	`rs2`	`rs1`	`funct3`	`rd`	`opcode`

add

`add rd, rs1, rs2`: R[rd] = R[rs1] + R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
`0000000`	`rs2`	`rs1`	`000`	`rd`	`0110011`

sub

`sub rd, rs1, rs2`: R[rd] = R[rs1] - R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
`0100000`	`rs2`	`rs1`	`000`	`rd`	`0110011`

and

`and rd, rs1, rs2`: R[rd] = R[rs1] & R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	111	rd	0110011

or

`or rd, rs1, rs2`: R[rd] = R[rs1] | R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	110	rd	0110011

xor

`xor rd, rs1, rs2`: R[rd] = R[rs1] ^ R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	100	rd	0110011

sll

`sll rd, rs1, rs2`: R[rd] = R[rs1] << R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	001	rd	0110011

sra

`sra rd, rs1, rs2`: R[rd] = R[rs1] >>s R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0100000	rs2	rs1	101	rd	0110011

srl

srl rd, rs1, rs2: R[rd] = R[rs1] >>u R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	101	rd	0110011

slt

slt rd, rs1, rs2: R[rd] = R[rs1] <s R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	010	rd	0110011

sltu

sltu rd, rs1, rs2: R[rd] = R[rs1] <u R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000000	rs2	rs1	011	rd	0110011

mul

mul rd, rs1, rs2: R[rd] = R[rs1] * R[rs2]

31-25	24-20	19-15	14-12	11-7	6-0
0000001	rs2	rs1	000	rd	0110011

mulh

`mulh rd, rs1, rs2`: (R[rd] = R[rs1] s*s R[rs2]) >> 32

31-25	24-20	19-15	14-12	11-7	6-0
0000001	rs2	rs1	001	rd	0110011

mulhu

`mulhu rd, rs1, rs2`: (R[rd] = R[rs1] u*u R[rs2]) >> 32

31-25	24-20	19-15	14-12	11-7	6-0
0000001	rs2	rs1	011	rd	0110011

csrrw

`csrrw rd, csr, rs1`: (t = CSRs[csr]; CSRs[csr] = R[rs1]; R[rd] = t)

31-20	19-15	14-12	11-7	6-0
csr	rs1	001	rd	1110011

I-Type

These instructions are similar but include an immediate value, or a literal constant as the second operand.

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	funct3	rd	opcode

addi

addi rd, rs1, immediate: R[rd] = R[rs1] + R[rs2]

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	000	rd	0010011

andi

andi rd, rs1, immediate: R[rd] = R[rs1] & R[rs2]

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	111	rd	0010011

ori

ori rd, rs1, immediate: R[rd] = R[rs1] | R[rs2]

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	110	rd	0010011

xori

xori rd, rs1, immediate: R[rd] = R[rs1] ^ R[rs2]

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	100	rd	0010011

slli

slli rd, rs1, immediate: R[rd] = R[rs1] << shamt

31-25	24-20	19-15	14-12	11-7	6-0
0000000	shamt[4:0]	rs1	001	rd	0010011

srai

srai rd, rs1, immediate: R[rd] = R[rs1] >>s shamt

31-25	24-20	19-15	14-12	11-7	6-0
0100000	shamt[4:0]	rs1	101	rd	0010011

srli

srli rd, rs1, immediate: R[rd] = R[rs1] >>u shamt

31-25	24-20	19-15	14-12	11-7	6-0
0000000	shamt[4:0]	rs1	101	rd	0010011

U-Type

These instructions are similar to I-type, but include a longer immediate value, and don't accept any source registers.

`lui`

`lui rd, immediate`: R[rd] = imm

Note that the lower 12 bits of the immediate value are treated as zero.

31-12	11-7	6-0
`imm[31:12]`	`rd`	`0110111`

Control and Status Register (CSR) Instructions

These instructions read and write to I/O registers. In this course, the IO registers will allow the CPU to interact with the switches, keys, and hex displays of the virtual DE2.

We will define a total of four I/Os, two inputs, and two outputs.

- `io0`, CSR 0xf00, input
- `io1`, CSR 0xf01, input
- `io2`, CSR 0xf02, output
- `io3`, CSR 0xf03, output

`io0` should be connected to the switches, and `io2` should be connected to the HEX displays. You may use `io1` and `io3` for whatever purpose you like, such as for debugging, connecting the LEDs or KEYs, etc.

Note: `io0`, `io1`, `io2`, and `io3` are not standard RISC-V CSRs. A specially modified version of RARS is included with your project skeleton that supports these CSRs. Official versions of MARS will not assemble programs which use these CSRs.

We only need to implement CSRRW for our purposes. Although it ostensibly both writes to and reads from a CSR, the CSRs that we implement are either read-only or write-only, so this instruction can be correctly used to read from or write to any of them. Reading from a write-only I/O such as `io2`, or from an unknown CSR address, should result in a value of `0`. Writing to a read-only I/O such as `io0`, or to an unknown CSR address, should have no effect.

`csrrw`

`csrrw rd, csr, rs1`: t = CSrs[csr]; CSrs[csr] = r[rs1]; r[rd] = t;

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	001	rd	1110011

RV32I I/J/B Instruction Reference

Note: In some cases, instructions are grouped with types other than their actual encoding for logical clarity. Such cases are specifically noted.

Note: All immediate values should be sign-extended. The sign bit of all immediate values is in the most-significant bit (bit 31) of the instruction.

I-Type

`jalr`

Note: this instruction deals with control-flow, but is encoded as an I-type.

`jalr rd, offset`: t=PC+4; PC += (R[rs1] + sext(offset))&~1; R[rd] = t

31-20	19-15	14-12	11-7	6-0
imm[11:0]	rs1	000	rd	1100111

B-Type

These instructions prevent the cpu from executing the next instruction in the program and instead cause it to begin executing a sequence of instructions in another memory location.

b

`b offset`: PC += sext(offset)

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

beq

`beq rs1, rs2, offset`: if (R[rs1]==R[rs2]) PC += sext(offset)

31	30-25	24-20	19-15	14-12	11-8	7	6-0
imm[12]	imm[10:5]	rs2	rs1	000	imm[4:1]	imm[11]	1100011

bge

`bge rs1, rs2, offset`: if (R[rs1]>=R[rs2]) PC += sext(offset)

31	30-25	24-20	19-15	14-12	11-8	7	6-0
imm[12]	imm[10:5]	rs2	rs1	101	imm[4:1]	imm[11]	1100011

bgeu

`bgeu rs1, rs2, offset`: if (R[rs1]>=uR[rs2]) PC += sext(offset)

31	30-25	24-20	19-15	14-12	11-8	7	6-0
imm[12]	imm[10:5]	rs2	rs1	101	imm[4:1]	imm[11]	1100011

blt

`blt rs1, rs2, offset`: if (R[rs1]<R[rs2]) PC += sext(offset)

31	30-25	24-20	19-15	14-12	11-8	7	6-0
imm[12]	imm[10:5]	rs2	rs1	100	imm[4:1]	imm[11]	1100011

bltu

bltu rs1, rs2, offset: if (R[rs1]<uR[rs2]) PC += sext(offset)

31	30-25	24-20	19-15	14-12	11-8	7	6-0
imm[12]	imm[10:5]	rs2	rs1	110	imm[4:1]	imm[11]	1100011

J-Type

Note: This instruction is the only J-type we will implement this semster.

jal

jal rd, offset: R[rd] = PC+4; PC += sext(offset)

31	30-21	20	19-12	11-7	6-0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	1101111

Full Instruction Table

For your convenience, the entire table of RV32I and M instructions is given below. Note that includes some instructions we will not use in this course.

mnemonic	spec	funct7	funct3	opcode	encoding
LUI	RV32I			0110111	U
AUIPC	RV32I			0010111	U
JAL	RV32I			1101111	J

mnemonic	spec	funct7	funct3	opcode	encoding
JALR	RV32I		000	1100111	I
BEQ	RV32I		000	1100011	B
BNE	RV32I		001	1100011	B
BLT	RV32I		100	1100011	B
BGE	RV32I		101	1100011	B
BLTU	RV32I		110	1100011	B
BGEU	RV32I		111	1100011	B
LB	RV32I		000	0000011	I
LH	RV32I		001	0000011	I
LW	RV32I		010	0000011	I
LBU	RV32I		100	0000011	I
LHU	RV32I		101	0000011	I
SB	RV32I		000	0100011	S
SH	RV32I		001	0100011	S
SW	RV32I		010	0100011	S
ADDI	RV32I		000	0010011	I
SLTI	RV32I		010	0010011	I
SLTIU	RV32I		011	0010011	I
XORI	RV32I		100	0010011	I
ORI	RV32I		110	0010011	I
ANDI	RV32I		111	0010011	I
SLLI	RV32I	0000000	001	0010011	R
SRLI	RV32I	0000000	101	0010011	R
SRAI	RV32I	0100000	101	0010011	R
ADD	RV32I	0000000	000	0110011	R
SUB	RV32I	0100000	000	0110011	R
SLL	RV32I	0000000	001	0110011	R

mnemonic	spec	funct7	funct3	opcode	encoding
SLT	RV32I	0000000	010	0110011	R
SLTU	RV32I	0000000	011	0110011	R
XOR	RV32I	0000000	100	0110011	R
SRL	RV32I	0000000	101	0110011	R
SRA	RV32I	0100000	101	0110011	R
OR	RV32I	0000000	110	0110011	R
AND	RV32I	0000000	111	0110011	R
FENCE	RV32I		000	0001111	I
FENCE.I	RV32I		001	0001111	I
ECALL	RV32I		000	1110011	I
EBREAK	RV32I		000	1110011	I
CSRRW	RV32I		001	1110011	I
CSRRS	RV32I		010	1110011	I
CSRRC	RV32I		011	1110011	I
CSRRWI	RV32I		101	1110011	I
CSRRSI	RV32I		110	1110011	I
CSRRCI	RV32I		111	1110011	I
MUL	RV32M	0000001	000	0110011	R
MULH	RV32M	0000001	001	0110011	R
MULHSU	RV32M	0000001	010	0110011	R
MULHU	RV32M	0000001	011	0110011	R
DIV	RV32M	0000001	100	0110011	R
DIVU	RV32M	0000001	101	0110011	R
REM	RV32M	0000001	110	0110011	R
REMU	RV32M	0000001	111	0110011	R

ALU Operation Table

This table is also given in the lecture slides, but is reproduced here for convenience.

Note that the ALU B input is used as the shift amount (shamt) for shift instructions, since it would otherwise be unused.

4-bit opcode	function
0000	A and B
0001	A or B
0010	A xor B
0011	A + B
0100	A - B
0101	A * B (low, signed)
0110	A * B (high, signed)
0111	A * B (high, unsigned)
1000	A << shamt
1001	A >> shamt
1010	A >>> shamt
1100	A < B (signed)
1101	A < B (unsigned)
1110	A < B (unsigned)
1111	A < B (unsigned)