MIT 6.191 (6.004) ISA Reference Card: Instructions

Instruction	Syntax	Description	Execution
LUI	lui rd, luiConstant	Load Upper Immediate	reg[rd] <= luiConstant « 12
JAL	jal rd, label	Jump and Link	reg[rd] <= pc + 4 pc <= label
JALR	jalr rd, offset(rs1)	Jump and Link Register	reg[rd] <= pc + 4 pc <= {(reg[rs1] + offset)[31:1], 1'b0}
BEQ	beq rs1, rs2, label	Branch if =	pc <= (reg[rs1] == reg[rs2]) ? label: pc + 4
BNE	bne rs1, rs2, label	Branch if ≠	pc <= (reg[rs1] != reg[rs2]) ? label: pc + 4
BLT	blt rs1, rs2, label	Branch if < (Signed)	pc <= (reg[rs1] <$_s$ reg[rs2]) ? label: pc + 4
BGE	bge rs1, rs2, label	Branch if ≥ (Signed)	pc <= (reg[rs1] >=$_s$ reg[rs2]) ? label: pc + 4
BLTU	bltu rs1, rs2, label	Branch if < (Unsigned)	pc <= (reg[rs1] <$_u$ reg[rs2]) ? label: pc + 4
BGEU	bgeu rs1, rs2, label	Branch if ≥ (Unsigned)	pc <= (reg[rs1] >=$_u$ reg[rs2]) ? label: pc + 4
LB	lb rd, offset(rs1)	Load Byte	reg[rd] <= signExtend(mem[addr])
LH	lh rd, offset(rs1)	Load Half Word	reg[rd] <= signExtend(mem[addr + 1: addr])
LW	lw rd, offset(rs1)	Load Word	reg[rd] <= mem[addr + 3: addr]
LBU	lbu rd, offset(rs1)	Load Byte (Unsigned)	reg[rd] <= zeroExtend(mem[addr])
LHU	lhu rd, offset(rs1)	Load Half Word (Unsigned)	reg[rd] <= zeroExtend(mem[addr + 1: addr])
SB	sb rs2, offset(rs1)	Store Byte	mem[addr] <= reg[rs2][7:0]
SH	sh rs2, offset(rs1)	Store Half Word	mem[addr + 1: addr] <= reg[rs2][15:0]
SW	sw rs2, offset(rs1)	Store Word	mem[addr + 3: addr] <= reg[rs2]
ADDI	addi rd, rs1, constant	Add Immediate	reg[rd] <= reg[rs1] + constant
SLTI	slti rd, rs1, constant	Compare < Immediate (Signed)	reg[rd] <= (reg[rs1] <$_s$ constant) ? 1 : 0
SLTIU	sltiu rd, rs1, constant	Compare < Immediate (Unsigned)	reg[rd] <= (reg[rs1] <$_u$ constant) ? 1 : 0
XORI	xori rd, rs1, constant	Xor Immediate	reg[rd] <= reg[rs1] ˆ constant
ORI	ori rd, rs1, constant	Or Immediate	reg[rd] <= reg[rs1] | constant
ANDI	andi rd, rs1, constant	And Immediate	reg[rd] <= reg[rs1] & constant
SLLI	slli rd, rs1, shamt	Shift Left Logical Immediate	reg[rd] <= reg[rs1] « shamt
SRLI	srli rd, rs1, shamt	Shift Right Logical Immediate	reg[rd] <= reg[rs1] »$_u$ shamt
SRAI	srai rd, rs1, shamt	Shift Right Arithmetic Immediate	reg[rd] <= reg[rs1] »$_s$ shamt
ADD	add rd, rs1, rs2	Add	reg[rd] <= reg[rs1] + reg[rs2]
SUB	sub rd, rs1, rs2	Subtract	reg[rd] <= reg[rs1] - reg[rs2]
SLL	sll rd, rs1, rs2	Shift Left Logical	reg[rd] <= reg[rs1] « reg[rs2][4:0]
SLT	slt rd, rs1, rs2	Compare < (Signed)	reg[rd] <= (reg[rs1] <$_s$ reg[rs2]) ? 1 : 0
SLTU	sltu rd, rs1, rs2	Compare < (Unsigned)	reg[rd] <= (reg[rs1] <$_u$ reg[rs2]) ? 1 : 0
XOR	xor rd, rs1, rs2	Xor	reg[rd] <= reg[rs1] ˆ reg[rs2]
SRL	srl rd, rs1, rs2	Shift Right Logical	reg[rd] <= reg[rs1] »$_u$ reg[rs2][4:0]
SRA	sra rd, rs1, rs2	Shift Right Arithmetic	reg[rd] <= reg[rs1] »$_s$ reg[rs2][4:0]
OR	or rd, rs1, rs2	Or	reg[rd] <= reg[rs1] | reg[rs2]
AND	and rd, rs1, rs2	And	reg[rd] <= reg[rs1] & reg[rs2]

Note:

- *luiConstant* is a 20-bit value.
- *offset* and *constant* are signed 12-bit values that are sign-extended to 32-bit values.
- *label* is a 32-bit memory address or its alias name.
- *shamt* is a 5-bit unsigned shift amount.
- *addr* is a load / store address as calculated by reg[rs1] + offset
- *addr* must also be aligned with the kind of data you are loading/storing
  - *addr* for **lh**, **lhu**, and **sh** must be divisible by 2
  - *addr* for **lw** and **sw** must be divisible by 4

MIT 6.191 (6.004) ISA Reference Card: Pseudoinstructions

Pseudoinstruction	Description	Execution
li rd, liConstant	Load Immediate	reg[rd] <= liConstant
mv rd, rs1	Move	reg[rd] <= reg[rs1] + 0
not rd, rs1	Logical Not	reg[rd] <= reg[rs1] ^ -1
neg rd, rs1	Arithmetic Negation	reg[rd] <= 0 - reg[rs1]
j label	Jump	pc <= label
jal label **call** label	Jump and Link (with ra)	reg[ra] <= pc + 4 pc <= label
jr rs1	Jump Register	pc <= reg[rs1] & ~1
jalr rs1	Jump and Link Register (with ra)	reg[ra] <= pc + 4 pc <= reg[rs1] & ~1
ret	Return from Subroutine	pc <= reg[ra]
bgt rs1, rs2, label	Branch $>$ (Signed)	pc <= (reg[rs1] $>_s$ reg[rs2]) ? label : pc + 4
ble rs1, rs2, label	Branch $\leq$ (Signed)	pc <= (reg[rs1] $<=_s$ reg[rs2]) ? label : pc + 4
bgtu rs1, rs2, label	Branch $>$ (Unsigned)	pc <= (reg[rs1] $>_s$ reg[rs2]) ? label : pc + 4
bleu rs1, rs2, label	Branch $\leq$ (Unsigned)	pc <= (reg[rs1] $<=_s$ reg[rs2]) ? label : pc + 4
beqz rs1, label	Branch $= 0$	pc <= (reg[rs1] == 0) ? label : pc + 4
bnez rs1, label	Branch $\neq 0$	pc <= (reg[rs1] != 0) ? label : pc + 4
bltz rs1, label	Branch < 0 (Signed)	pc <= (reg[rs1] $<_s$ 0) ? label : pc + 4
bgez rs1, label	Branch ≥ 0 (Signed)	pc <= (reg[rs1] $>=_s$ 0) ? label : pc + 4
bgtz rs1, label	Branch > 0 (Signed)	pc <= (reg[rs1] $>_s$ 0) ? label : pc + 4
blez rs1, label	Branch ≤ 0 (Signed)	pc <= (reg[rs1] $<=_s$ 0) ? label : pc + 4

Note: *liConstant* is a 32-bit value.

MIT 6.191 (6.004) ISA Reference Card: Calling Convention

Registers	Symbolic names	Description	Saver
x0	zero	Hardwired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporary registers	Caller
x8-x9	s0-s1	Saved registers	Callee
x10-x11	a0-a1	Function arguments and return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporary registers	Caller

MIT 6.191 (6.004) ISA Reference Card: Instruction Encodings

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

RV32I Base Instruction Set (MIT 6.191 (6.004) subset)

~~imm[31:12]~~				~~rd~~	~~0110111~~	~~LUI~~
~~imm[20|10:1|11|19:12]~~				~~rd~~	~~1101111~~	~~JAL~~
~~imm[11:0]~~		~~rs1~~	~~000~~	~~rd~~	~~1100111~~	~~JALR~~
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
~~imm[11:0]~~		~~rs1~~	~~000~~	~~rd~~	~~0000011~~	~~LB~~
~~imm[11:0]~~		~~rs1~~	~~001~~	~~rd~~	~~0000011~~	~~LH~~
imm[11:0]		rs1	010	rd	0000011	LW
~~imm[11:0]~~		~~rs1~~	~~100~~	~~rd~~	~~0000011~~	~~LBU~~
~~imm[11:0]~~		~~rs1~~	~~101~~	~~rd~~	~~0000011~~	~~LHU~~
~~imm[11:5]~~	~~rs2~~	~~rs1~~	~~000~~	~~imm[4:0]~~	~~0100011~~	~~SB~~
~~imm[11:5]~~	~~rs2~~	~~rs1~~	~~001~~	~~imm[4:0]~~	~~0100011~~	~~SH~~
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

- For JAL and branch instructions (BEQ, BNE, BLT, BGE, BLTU, BGEU), the immediate encodes the target address as an offset from the current pc (i.e., $pc + imm = label$).
- Not all immediate bits are encoded. Missing lower bits are filled with zeros and missing upper bits are sign-extended.