RV32I, RV64I Instructions

lui

load upper immediate.

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[31:12]					rd	01101	11

Format:	lui rd,imm
Description:	Build 32-bit constants and uses the U-type format. LUI places the U-immediate value in the top 20 bits of the destination register rd, filling in the lowest 12 bits with zeros.
Implementation:	x[rd] = sext(immediate[31:12] << 12)

auipc

add upper immediate to pc

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[31:12]					rd	00101	11

Format:	auipc rd,imm
Description:	Build pc-relative addresses and uses the U-type format. AUIPC forms a 32-bit offset from the 20-bit U-immediate, filling in the lowest 12 bits with zeros, adds this offset to the pc, then places the result in register rd.
Implementation:	x[rd] = pc + sext(immediate[31:12] << 12)

addi

add immediate

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0

imm[11:0]			rs1	000	rd	00100	11

Format:	addi rd,rs1,imm
Description:	Adds the sign-extended 12-bit immediate to register rs1. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result. ADDI rd, rs1, 0 is used to implement the MV rd, rs1 assembler pseudo-instruction.
Implementation:	x[rd] = x[rs1] + sext(immediate)

slti

set less than immediate

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	010	rd	00100	11

Format:	slti rd,rs1,imm
Description:	Place the value 1 in register rd if register rs1 is less than the signextended immediate when both are treated as signed numbers, else 0 is written to rd.
Implementation:	x[rd] = x[rs1] <s sext(immediate)

sltiu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	011	rd	00100	11

Format:	sltiu rd,rs1,imm
Description:	Place the value 1 in register rd if register rs1 is less than the immediate when both are treated as unsigned numbers, else 0 is written to rd.
Implementation:	x[rd] = x[rs1] <u sext(immediate)

xori

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	100	rd	00100	11

Format: xori rd,rs1,imm

Description: Performs bitwise XOR on register rs1 and the sign-extended 12-bit immediate and place the result in rd
Note, “XORI rd, rs1, -1” performs a bitwise logical inversion of register rs1(assembler pseudo-instruction NOT rd, rs)

Implementation: x[rd] = x[rs1] ^ sext(immediate)

ori

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	110	rd	00100	11

Format: ori rd,rs1,imm

Description: Performs bitwise OR on register rs1 and the sign-extended 12-bit immediate and place the result in rd

Implementation: x[rd] = x[rs1] | sext(immediate)

andi

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	111	rd	00100	11

Format: andi rd,rs1,imm

Description: Performs bitwise AND on register rs1 and the sign-extended 12-bit immediate and place the result in rd

Implementation: x[rd] = x[rs1] & sext(immediate)

slli

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	0X	shamt	rs1	001	rd	00100	11

Format: slli rd,rs1,shamt

Description: Performs logical left shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate
In RV64, bit-25 is used to shamt[5].

Implementation: x[rd] = x[rs1] << shamt

srli

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	0X	shamt	rs1	101	rd	00100	11

Format: srli rd,rs1,shamt

Description: Performs logical right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate
In RV64, bit-25 is used to shamt[5].

Implementation: x[rd] = x[rs1] >>u shamt

srai

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	0X	shamt	rs1	101	rd	00100	11

Format: srai rd,rs1,shamt

Description: Performs arithmetic right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate
In RV64, bit-25 is used to shamt[5].

Implementation: x[rd] = x[rs1] >>s shamt

add

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	000	rd	01100	11

Format: add rd,rs1,rs2

Description: Adds the registers rs1 and rs2 and stores the result in rd.
Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result.

Implementation: x[rd] = x[rs1] + x[rs2]

sub

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	00	rs2	rs1	000	rd	01100	11

Format: sub rd,rs1,rs2

Description: Subs the register rs2 from rs1 and stores the result in rd.
Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result.

Implementation: x[rd] = x[rs1] - x[rs2]

sll

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	001	rd	01100	11

Format: sll rd,rs1,rs2

Description: Performs logical left shift on the value in register rs1 by the shift amount held in the lower 5 bits of register rs2.

Implementation: x[rd] = x[rs1] << x[rs2]

slt

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	010	rd	01100	11

Format: slt rd,rs1,rs2

Description: Place the value 1 in register rd if register rs1 is less than register rs2 when both are treated as signed numbers, else 0 is written to rd.

Implementation: x[rd] = x[rs1] <s x[rs2]

sltu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	011	rd	01100	11

Format:	sltu rd,rs1,rs2
Description:	Place the value 1 in register rd if register rs1 is less than register rs2 when both are treated as unsigned numbers, else 0 is written to rd.
Implementation:	x[rd] = x[rs1] <u x[rs2]

xor

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	100	rd	01100	11

Format:	xor rd,rs1,rs2
Description:	Performs bitwise XOR on registers rs1 and rs2 and place the result in rd
Implementation:	x[rd] = x[rs1] ^ x[rs2]

srl

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	101	rd	01100	11

Format:	srl rd,rs1,rs2
Description:	Logical right shift on the value in register rs1 by the shift amount held in the lower 5 bits of register rs2
Implementation:	x[rd] = x[rs1] >>u x[rs2]

sra

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	00	rs2	rs1	101	rd	01100	11

Format: sra rd,rs1,rs2

Description: Performs arithmetic right shift on the value in register rs1 by the shift amount held in the lower 5 bits of register rs2

Implementation: x[rd] = x[rs1] >>s x[rs2]

or

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	110	rd	01100	11

Format: or rd,rs1,rs2

Description: Performs bitwise OR on registers rs1 and rs2 and place the result in rd

Implementation: x[rd] = x[rs1] | x[rs2]

and

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	111	rd	01100	11

Format: and rd,rs1,rs2

Description: Performs bitwise AND on registers rs1 and rs2 and place the result in rd

Implementation: x[rd] = x[rs1] & x[rs2]

fence

31-28	27-24	23-20	19-15	14-12	11-7	6-2	1-0
0000	pred	succ	00000	000	00000	00011	11

Format: fence pred, succ

Description: Used to order device I/O and memory accesses as viewed by other RISC-V harts and external devices or coprocessors.
Any combination of device input (I), device output (O), memory reads (R), and memory writes (W) may be ordered with respect to any combination of the same.
Informally, no other RISC-V hart or external device can observe any operation in the successor set following a FENCE before any operation in the predecessor set preceding the FENCE.

Implementation: Fence(pred, succ)

fence.i

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	00000	00000	001	00000	00011	11

Format: fence.i

Description: Provides explicit synchronization between writes to instruction memory and instruction fetches on the same hart.

Implementation: Fence(Store, Fetch)

csrrw

atomic read/write CSR.

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			rs1	001	rd	11100	11

Format: csrrw rd,offset,rs1

Description: Atomically swaps values in the CSRs and integer registers.
CSRRW reads the old value of the CSR, zero-extends the value to XLEN bits, then writes it to integer register rd.
The initial value in rs1 is written to the CSR.
If rd=x0, then the instruction shall not read the CSR and shall not cause any of the side effects that might occur on a CSR read.

Implementation: t = CSRs[csr]; CSRs[csr] = x[rs1]; x[rd] = t

csrrs

atomic read and set bits in CSR.

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			rs1	010	rd	11100	11

Format:	csrrs rd,offset,rs1
Description:	Reads the value of the CSR, zero-extends the value to XLEN bits, and writes it to integer register rd. The initial value in integer register rs1 is treated as a bit mask that specifies bit positions to be set in the CSR. Any bit that is high in rs1 will cause the corresponding bit to be set in the CSR, if that CSR bit is writable. Other bits in the CSR are unaffected (though CSRs might have side effects when written).
Implementation:	t = CSRs[csr]; CSRs[csr] = t | x[rs1]; x[rd] = t

csrrc

atomic read and clear bits in CSR.

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			rs1	011	rd	11100	11

Format:	csrrc rd,offset,rs1
Description:	Reads the value of the CSR, zero-extends the value to XLEN bits, and writes it to integer register rd. The initial value in integer register rs1 is treated as a bit mask that specifies bit positions to be cleared in the CSR. Any bit that is high in rs1 will cause the corresponding bit to be cleared in the CSR, if that CSR bit is writable. Other bits in the CSR are unaffected.
Implementation:	t = CSRs[csr]; CSRs[csr] = t &~x[rs1]; x[rd] = t

csrrwi

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			uimm	101	rd	11100	11

Format:	csrrwi rd,offset,uimm
Description:	Update the CSR using an XLEN-bit value obtained by zero-extending a 5-bit unsigned immediate (uimm[4:0]) field encoded in the rs1 field.
Implementation:	x[rd] = CSRs[csr]; CSRs[csr] = zimm

csrrsi

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			uimm	110	rd	11100	11

Format:	csrrsi rd,offset,uimm
Description:	Set CSR bit using an XLEN-bit value obtained by zero-extending a 5-bit unsigned immediate (uimm[4:0]) field encoded in the rs1 field.
Implementation:	t = CSRs[csr]; CSRs[csr] = t | zimm; x[rd] = t

csrrci

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
csr			rs1	111	rd	11100	11

Format:	csrrci rd,offset,uimm
Description:	Clear CSR bit using an XLEN-bit value obtained by zero-extending a 5-bit unsigned immediate (uimm[4:0]) field encoded in the rs1 field.
Implementation:	t = CSRs[csr]; CSRs[csr] = t &∼zimm; x[rd] = t

ecall

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0

00000	00	00000	00000	000	00000	11100	11

Format:	ecall
Description:	Make a request to the supporting execution environment. When executed in U-mode, S-mode, or M-mode, it generates an environment-call-from-U-mode exception, environment-call-from-S-mode exception, or environment-call-from-M-mode exception, respectively, and performs no other operation.
Implementation:	RaiseException(EnvironmentCall)

ebreak

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	00001	00000	000	00000	11100	11

Format:	ebreak
Description:	Used by debuggers to cause control to be transferred back to a debugging environment. It generates a breakpoint exception and performs no other operation.
Implementation:	RaiseException(Breakpoint)

uret

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	00010	00000	000	00000	11100	11

Format:	uret
Description:	Return from traps in U-mode, and URET copies UPIE into UIE, then sets UPIE.
Implementation:	ExceptionReturn(User)

sret

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00010	00	00010	00000	000	00000	11100	11

Format:	sret
Description:	Return from traps in S-mode, and SRET copies SPIE into SIE, then sets SPIE.
Implementation:	ExceptionReturn(User)

mret

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00110	00	00010	00000	000	00000	11100	11

Format:	mret
Description:	Return from traps in M-mode, and MRET copies MPIE into MIE, then sets MPIE.
Implementation:	ExceptionReturn(Machine)

wfi

wait for interrupt.

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00010	00	00101	00000	000	00000	11100	11

Format:	wfi
Description:	Provides a hint to the implementation that the current hart can be stalled until an interrupt might need servicing. Execution of the WFI instruction can also be used to inform the hardware platform that suitable interrupts should preferentially be routed to this hart. WFI is available in all privileged modes, and optionally available to U-mode. This instruction may raise an illegal instruction exception when TW=1 in mstatus.
Implementation:	while (noInterruptsPending) idle

sfence.vma

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00010	01	rs2	rs1	000	rd	11100	11

Format: sfence.vma rs1,rs2

Description: Guarantees that any previous stores already visible to the current RISC-V hart are ordered before all subsequent implicit references from that hart to the memory-management data structures.
The SFENCE.VMA is used to flush any local hardware caches related to address translation.
It is specified as a fence rather than a TLB flush to provide cleaner semantics with respect to which instructions are affected by the flush operation and to support a wider variety of dynamic caching structures and memory-management schemes.
SFENCE.VMA is also used by higher privilege levels to synchronize page table writes and the address translation hardware.

Implementation: Fence(Store, AddressTranslation)

lb

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	000	rd	00000	11

Format: lb rd,offset(rs1)

Description: Loads a 8-bit value from memory and sign-extends this to XLEN bits before storing it in register rd.

Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][7:0])

lh

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	001	rd	00000	11

Format: lh rd,offset(rs1)

Description: Loads a 16-bit value from memory and sign-extends this to XLEN bits before storing it in register rd.

Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][15:0])

lw

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	010	rd	00000	11

Format: lw rd,offset(rs1)

Description: Loads a 32-bit value from memory and sign-extends this to XLEN bits before storing it in register rd.

Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][31:0])

lbu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	100	rd	00000	11

Format: lbu rd,offset(rs1)

Description: Loads a 8-bit value from memory and zero-extends this to XLEN bits before storing it in register rd.

Implementation: x[rd] = M[x[rs1] + sext(offset)][7:0]

lhu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	101	rd	00000	11

Format: lhu rd,offset(rs1)

Description: Loads a 16-bit value from memory and zero-extends this to XLEN bits before storing it in register rd.

Implementation: x[rd] = M[x[rs1] + sext(offset)][15:0]

sb

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0

offset[11:5]		rs2	rs1	000	offset[4:0]	01000	11

Format: sb rs2,offset(rs1)

Description: Store 8-bit, values from the low bits of register rs2 to memory.

Implementation: M[x[rs1] + sext(offset)] = x[rs2][7:0]

sh

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:5]		rs2	rs1	001	offset[4:0]	01000	11

Format: sh rs2,offset(rs1)

Description: Store 16-bit, values from the low bits of register rs2 to memory.

Implementation: M[x[rs1] + sext(offset)] = x[rs2][15:0]

sw

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:5]		rs2	rs1	010	offset[4:0]	01000	11

Format: sw rs2,offset(rs1)

Description: Store 32-bit, values from the low bits of register rs2 to memory.

Implementation: M[x[rs1] + sext(offset)] = x[rs2][31:0]

jal

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[20|10:1|11			19:12]		rd	11011	11

Format: jal rd,offset

Description: Jump to address and place return address in rd.

Implementation: x[rd] = pc+4; pc += sext(offset)

jalr

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	000	rd	11001	11

Format: jalr rd,rs1,offset

Description: Jump to address and place return address in rd.

Implementation: t =pc+4; pc=(x[rs1]+sext(offset))&∼1; x[rd]=t

beq

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	000	offset[4:1|11]	11000	11

Format: beq rs1,rs2,offset

Description: Take the branch if registers rs1 and rs2 are equal.

Implementation: if (x[rs1] == x[rs2]) pc += sext(offset)

bne

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	001	offset[4:1|11]	11000	11

Format: bne rs1,rs2,offset

Description: Take the branch if registers rs1 and rs2 are not equal.

Implementation: if (x[rs1] != x[rs2]) pc += sext(offset)

blt

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	100	offset[4:1|11]	11000	11

Format: blt rs1,rs2,offset

Description:	Take the branch if registers rs1 is less than rs2, using signed comparison.
Implementation:	if (x[rs1] <s x[rs2]) pc += sext(offset)

bge

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	101	offset[4:1|11]	11000	11

Format:	bge rs1,rs2,offset
Description:	Take the branch if registers rs1 is greater than or equal to rs2, using signed comparison.
Implementation:	if (x[rs1] >=s x[rs2]) pc += sext(offset)

bltu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	110	offset[4:1|11]	11000	11

Format:	bltu rs1,rs2,offset
Description:	Take the branch if registers rs1 is less than rs2, using unsigned comparison.
Implementation:	if (x[rs1] <u x[rs2]) pc += sext(offset)

bgeu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	111	offset[4:1|11]	11000	11

Format:	bgeu rs1,rs2,offset
Description:	Take the branch if registers rs1 is greater than or equal to rs2, using unsigned comparison.
Implementation:	if (x[rs1] >=u x[rs2]) pc += sext(offset)