

RV32I INSTRUCTION SET ARCHITECTURE REFERENCE CARD

Muhammed Erkmen & Oğuzhan Vatansever

zxt: zero extend, sxt: sign extend, PC: program counter

LUI

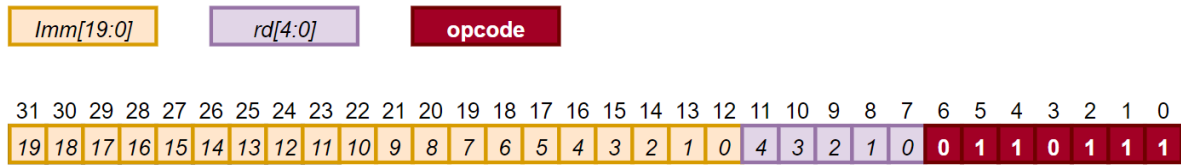


Figure 1. LUI Instruction Bit sequence

Load Upper Immediate instruction shifts Immediate value 12 bits left, fills that bits with 1'b0 and loads that value to rd.

$$rd = (Imm \ll 12)$$

AUIPC

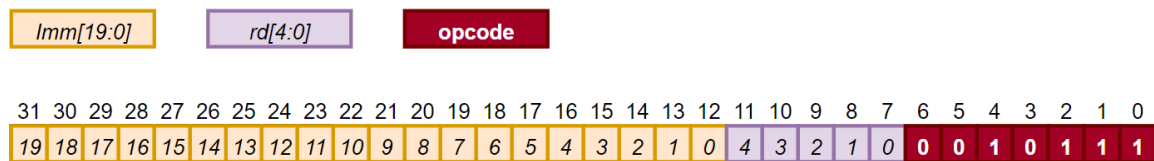


Figure 2. AUIPC Instruction Bit sequence

Add Upper Immediate to PC instruction shift immediate value 12 bits left, fills that bits with 1'b0 and adds that value with PC (program counter) value and loads that value to rd.

$$rd = PC + (Imm \ll 12)$$

BEQ

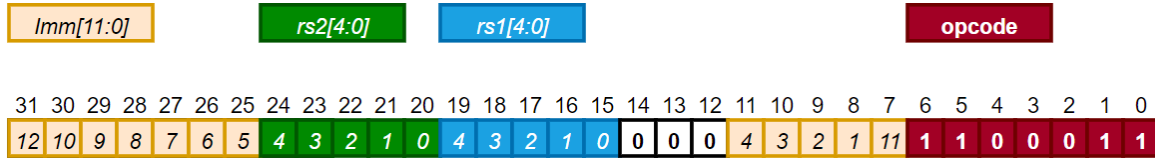


Figure 3. BEQ Instruction Bit sequence

BEQ (branch if equal) instruction compares rs1 and rs2. If they are equal, adds PC and Immediate values and loads result to PC. If they are not equal, it loads PC +4 to PC.

$$PC = (rs1 == rs2) ? (PC + Imm) : (PC + 4)$$

BNE

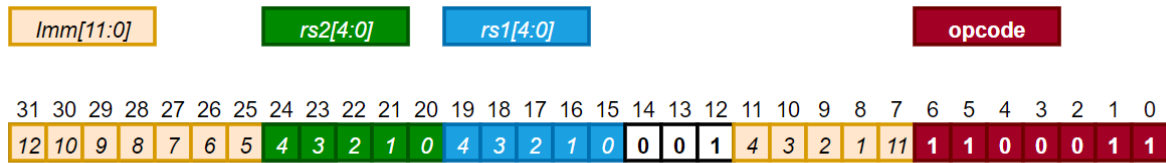


Figure 4. BNE Instruction Bit sequence

BNE (branch not equal) instruction compares rs1 and rs2. If they are not equal, adds PC and Immediate values and loads result to PC. If they are equal, it loads PC+4 to PC.

$$PC = (rs1 \neq rs2) ? (PC + Imm) : (PC + 4)$$

BLT

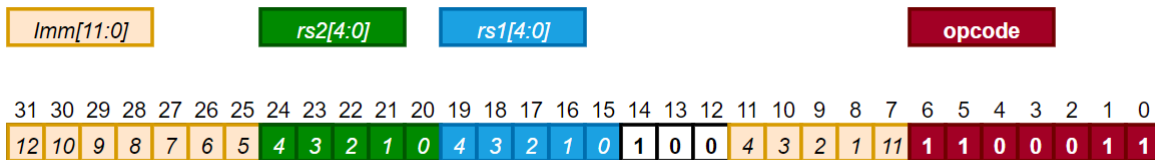


Figure 5. BLT Instruction Bit sequence

BLT (branch less than) instruction compares signed rs1 and rs2. If rs1 is less than rs2, adds PC and Immediate values and loads the result to PC. Else loads PC+4 to PC.

$$PC = (\text{signed}(rs1) < \text{signed}(rs2)) ? (PC + Imm) : (PC + 4)$$

BGE

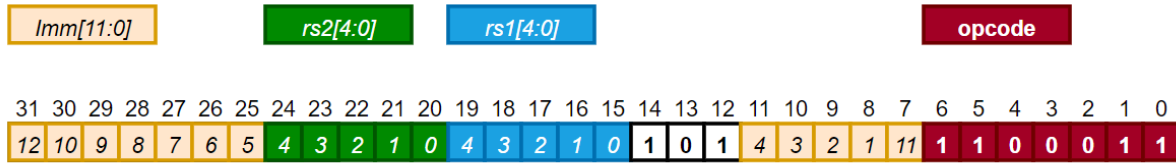


Figure 6. BGE Instruction Bit sequence

BGE (branch greater equal) instruction compares signed rs1 and rs2. If rs1 is greater or equal to rs2, adds PC and Immediate values and loads the result to PC. Else loads PC+4 to PC.

$$PC = (\text{signed}(rs1) \geq \text{signed}(rs2)) ? (PC + Imm) : (PC + 4)$$

BLTU

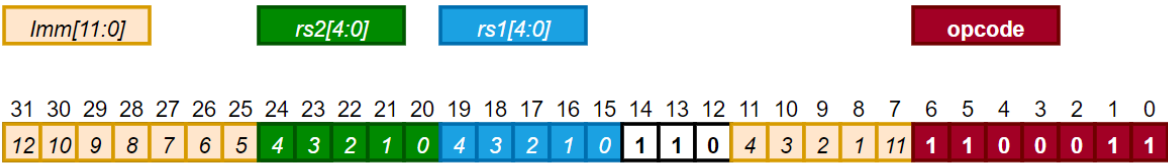


Figure 7. BLTU Instruction Bit sequence

BLTU (branch less than unsigned) instruction compares unsigned rs1 and rs2. If rs1 is less than rs2, adds PC and Immediate values and loads the result to PC. Else loads PC+4 to PC.

$$PC = (\text{unsigned}(rs1) < \text{unsigned}(rs2)) ? (PC + Imm) : (PC + 4)$$

BGEU

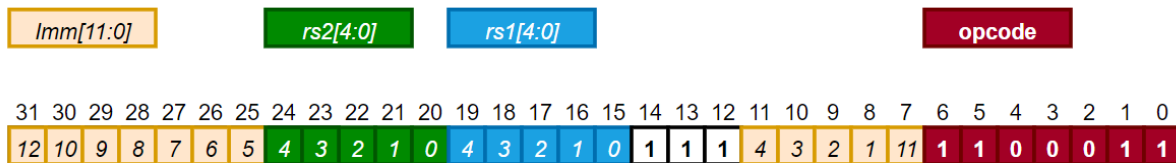


Figure 8. BGEU Instruction Bit sequence

BGEU (branch greater or equal unsigned) instruction compares unsigned rs1 and rs2. If rs1 is greater than or equal to rs2, adds PC and Immediate values and loads the result to PC. Else loads PC+4 to PC.

$$PC = (\text{unsigned}(rs1) \geq \text{unsigned}(rs2)) ? (PC + Imm) : (PC + 4)$$

LB

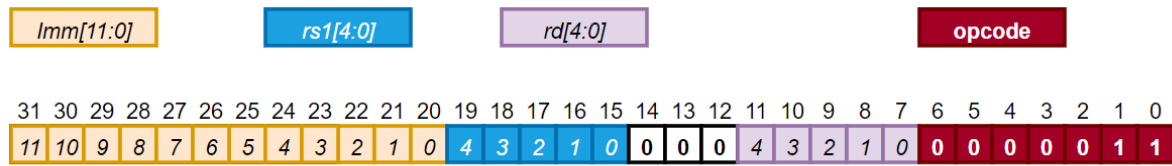


Figure 9. LB Instruction Bit sequence

LB (Load Byte) instruction loads the signed byte value in memory of the address which obtained from signed addition rs1 and Imm, to rd.

$$rd = sxt(Mem[rs1 + Imm][7:0])$$

LH

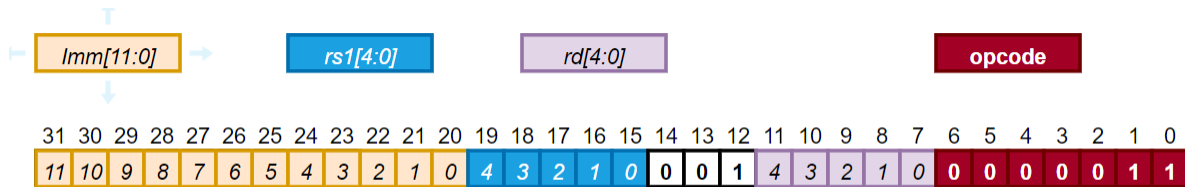


Figure 10. LH Instruction Bit sequence

LH (load halfword) instruction loads the signed halfword value in memory of the address, which obtained from signed addition rs1 and Imm, to rd.

$$rd = sxt(Mem[rs1 + Imm][15:0])$$

LW

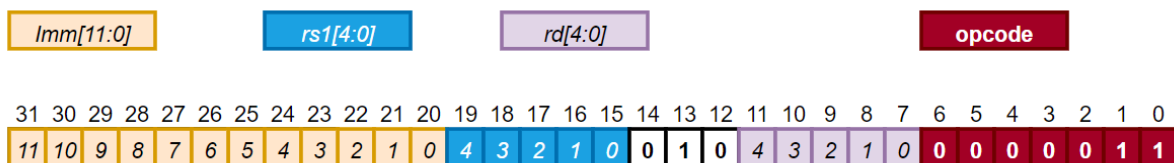


Figure 11. LW Instruction Bit sequence

LW (load word) instruction loads 4 byte value in memory address, which obtained from signed addition rs1 and Imm, to rd.

$$rd = Mem[rs1 + Imm][31:0]$$

LBU

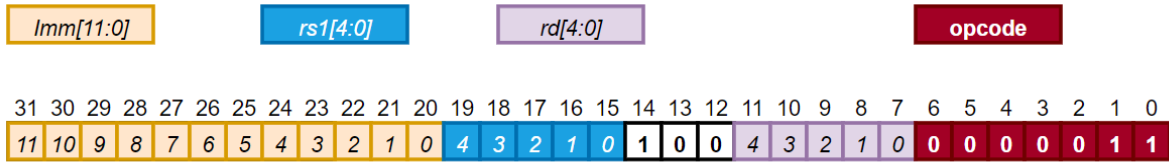


Figure 12. LBU Instruction Bit sequence

LBU (load byte unsigned) instruction loads the unsigned byte value, which is obtained from signed addition of *rs1* and *Imm*.

$$rd = zxt(Mem[rs1 + Imm][7:0])$$

LHU

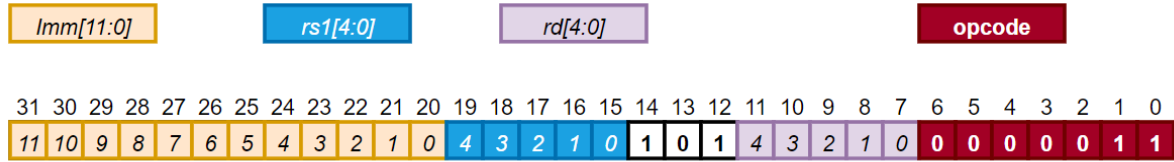


Figure 13. LHU Instruction Bit sequence

LHU (load halfword unsigned) instruction loads the unsigned halfword value, which is obtained from signed addition of *rs1* and *Imm*.

$$rd = zxt(Mem[rs1 + Imm][15:0])$$

SB

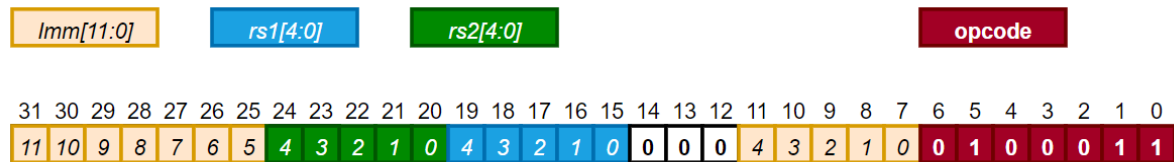


Figure 14. SB Instruction Bit sequence

SB (store byte) instruction stores a desired byte in *rs2*, to memory address which is obtained by signed addition of *rs1* and *Imm*.

$$Mem[rs1 + Imm][7:0] = rs2[7:0]$$

SH

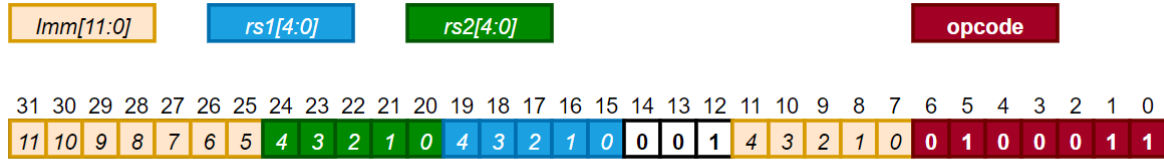


Figure 15. SH Instruction Bit sequence

SH (store halfword) instruction stores the desired 2 byte in *rs2*, to memory address which is obtained by signed addition of *rs1* and *Imm*.

$$Mem[rs1 + Imm][15:0] = rs2[15:0]$$

SW

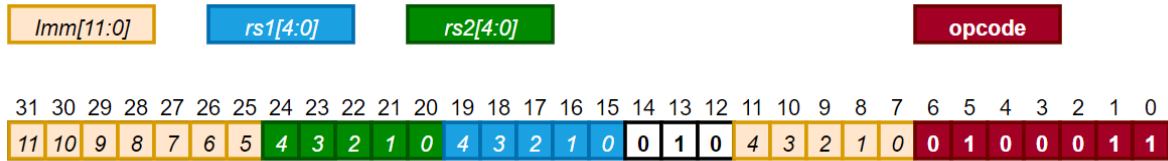


Figure 16. SW instruction bit sequence.

SW (store word) instruction stores the word in *rs2*, to memory address which is obtained by signed addition of *rs1* and *Imm*.

$$Mem[rs1 + Imm][31:0] = rs2[31:0]$$

ADDI

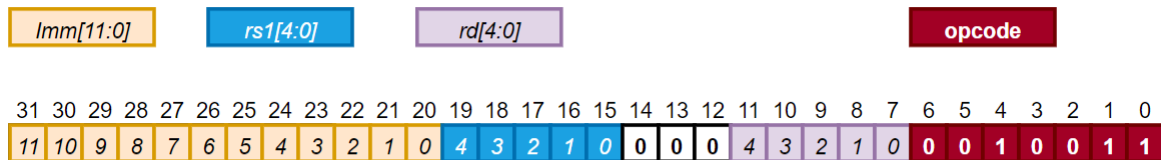


Figure 17. ADDI instruction bit sequence.

ADDI (Add immediate) instruction adds *rs1* and sign extended immediate value. Loads the result value to *rd*.

$$rd = rs1 + sxt(Imm)$$

SLTI

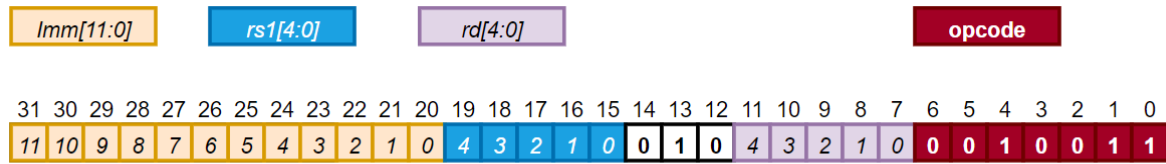


Figure 18. SLTI instruction bit sequence.

SLTI (set less than immediate) instruction compares signed rs1 and sign extended signed immediate values. If signed rs1 is less than sign extended signed immediate value, loads rd to decimal 1, else loads rd to decimal 0.

$$rd = \text{signed}(rs1) < \text{signed}(sxt(Imm)) ? 1 : 0$$

SLTIU

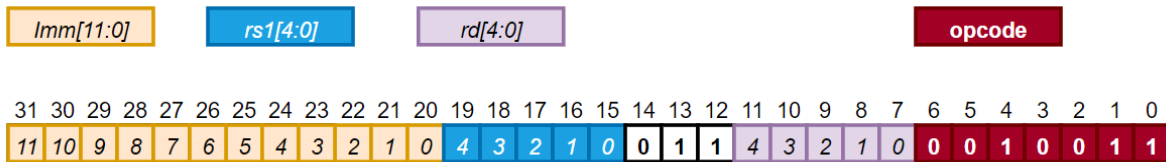


Figure 19. SLTIU instruction bit sequence.

SLTIU (set less than unsigned immediate) instruction compares unsigned rs1 and sign extended unsigned immediate values. If unsigned rs1 is less than sign extended unsigned immediate value, loads rd to decimal 1, else loads rd to decimal 0.

$$rd = \text{unsigned}(rs1) < \text{unsigned}(sxt(Imm)) ? 1 : 0$$

XORI

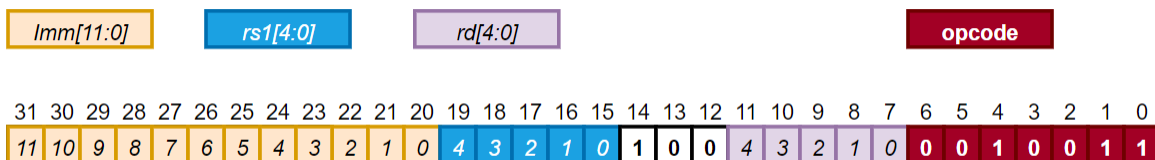


Figure 20. XORI instruction bit sequence.

XORI (logical XOR with immediate) instruction makes logical operation rs1 XOR sign extended immediate and loads the result to rd.

$$rd = rs1 \wedge sxt(Imm)$$

ORI

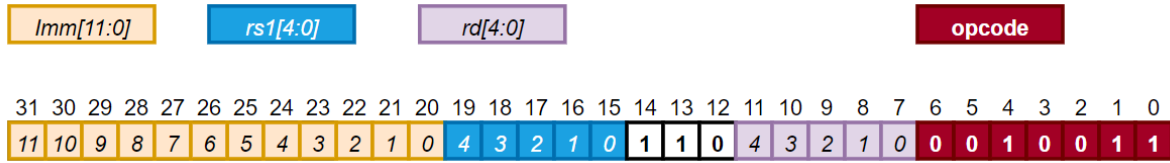


Figure 21. ORI instruction bit sequence.

ORI (logical OR with immediate) instruction makes logical operation $rs1$ OR sign extended immediate and loads the result to rd .

$$rd = rs1 \mid sxt(Imm)$$

ANDI

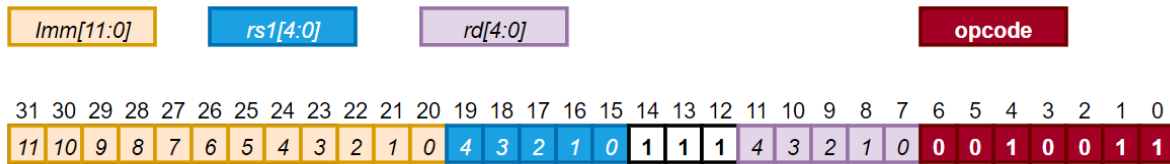


Figure 22. ANDI instruction bit sequence.

ANDI (logical AND with immediate) instruction makes logical operation $rs1$ AND sign extended immediate and loads the result to rd .

$$rd = rs1 \& sxt(Imm)$$

SLLI

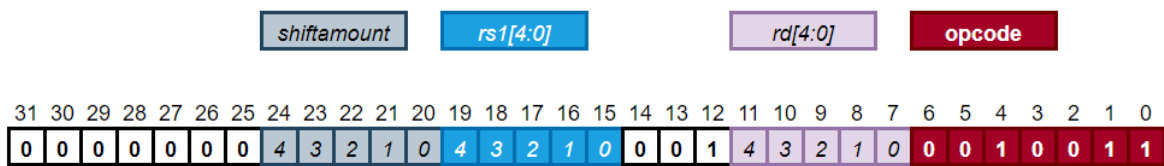


Figure 23. SLLI Instruction Bit sequence

SLLI (logical left shift by immediate) instruction shifts $rs1$ left by immediate value(shiftamount) and loads the result to rd .

$$rd = rs1 \ll Imm (shiftamount)$$

SRLI

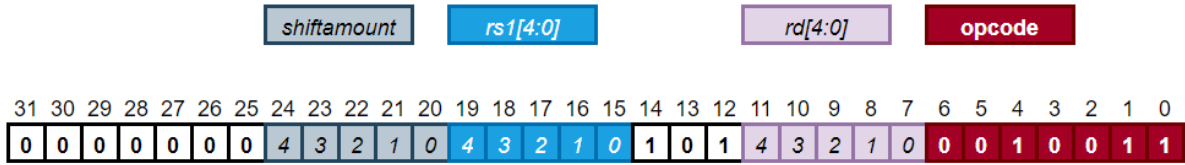


Figure 24. SRLI Instruction Bit sequence

SRLI (logical right shift by immediate) instruction shifts rs1 right by immediate value(shiftamount) and loads the result to rd.

$$rd = rs1 \gg Imm(shiftamount)$$

SRAI

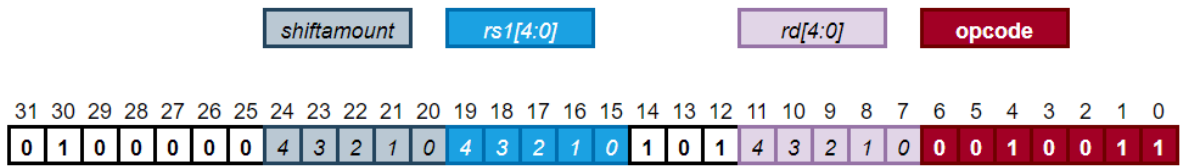


Figure 25. SRAI Instruction Bit sequence

SRAI (arithmetic right shift by immediate) instruction shifts rs1 right by immediate value(shiftamount), sign bit of rs1 get copied into upper bits and loads the result to rd.

$$rd = rs1 >>> Imm(shiftamount)$$

ADD

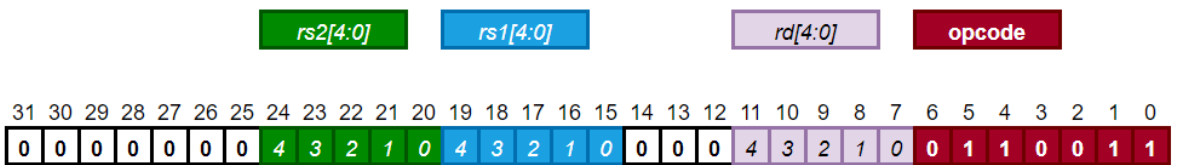


Figure 26. ADD Instruction Bit sequence

ADD (addition) instruction adds rs1 and rs2 and loads the result to rd.

$$rd = rs1 + rs2$$

SUB

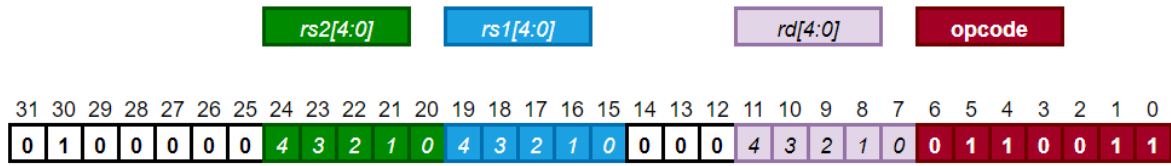


Figure 27. SUB Instruction Bit sequence

SUB (subtraction) instruction subtracts rs2 from rs2 and loads the result to rd.

$$rd = rs1 - rs2$$

SLL

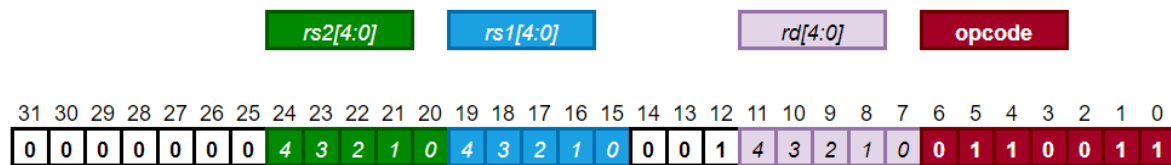


Figure 28. SLL Instruction Bit sequence

SLL (shift left logical) instruction shifts rs1 left by mod32(rs2) times and loads the result to rd.

$$rd = rs1 \ll rs2[4:0]$$

SLT

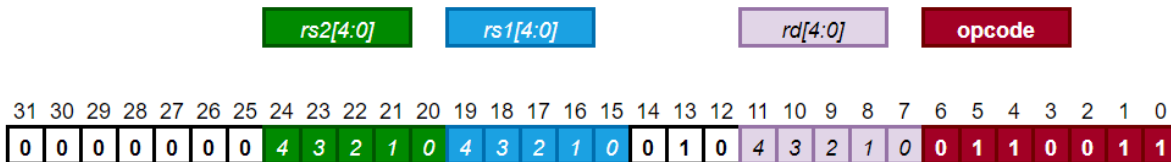


Figure 29. SLT Instruction Bit sequence

SLT (set less than) instruction compares signed rs1 and signed rs2. If signed rs1 is less than signed rs2, loads rd to decimal 1; else loads rd to decimal 0.

$$rd = \text{signed}(rs1) < \text{signed}(rs2) ? 1 : 0$$

SLTU

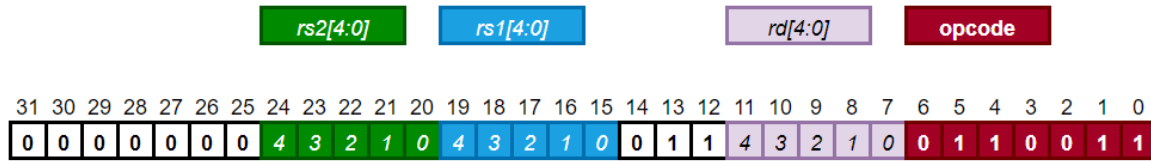


Figure 30. SLTU Instruction Bit sequence

SLTU (set less than unsigned) instruction compares unsigned rs1 and unsigned rs2. If unsigned rs1 is less than unsigned rs2, loads rd to decimal 1; else loads rd to decimal 0.

$$rd = \text{unsigned}(rs1) < \text{unsigned}(rs2) ? 1 : 0$$

XOR

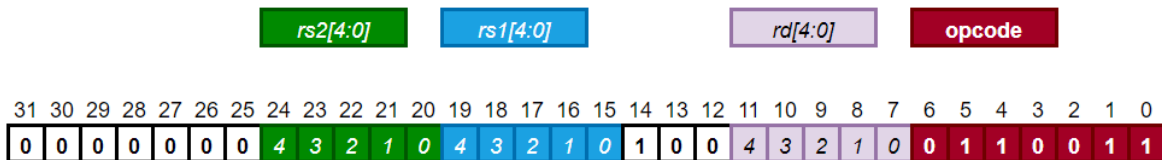


Figure 31. XOR Instruction Bit sequence

XOR (Logical XOR) instruction makes operation rs1 XOR rs2 and loads the result to rd.

SRL



Figure 32. SRL Instruction Bit sequence

SRL (shift right logical) instruction shifts rs1 right by mod32(rs2) times.

$$rd = rs1 \gg rs2[4:0]$$

SRA

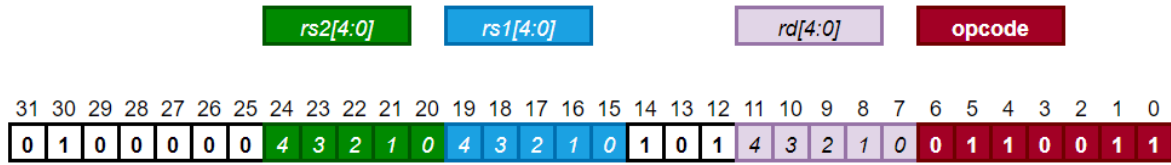


Figure 33. SRA Instruction Bit sequence

SRA (shift right arithmetic) instruction shifts *rs1* right by $\text{mod}32(\text{rs2})$ times and copies sign bit of *rs1* into upper bits, loads the result to *rd*.

$$rd = rs1 \gg rs2[4:0]$$

OR

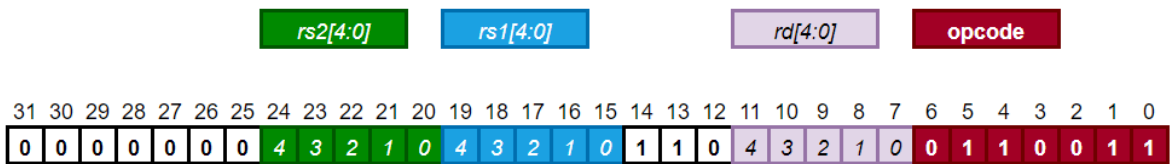


Figure 34. OR Instruction Bit sequence

OR (logical OR) instruction makes *rs1* OR *rs2* operation and loads the result to *rd*.

$$rd = rs1 | rs2$$

AND

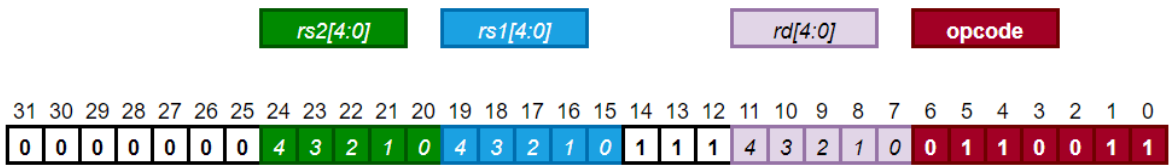


Figure 35. AND Instruction Bit sequence

AND (logical AND) instruction makes *rs1* AND *rs2* operation and loads the result to *rd*.

$$rd = rs1 \& rs2$$

JAL

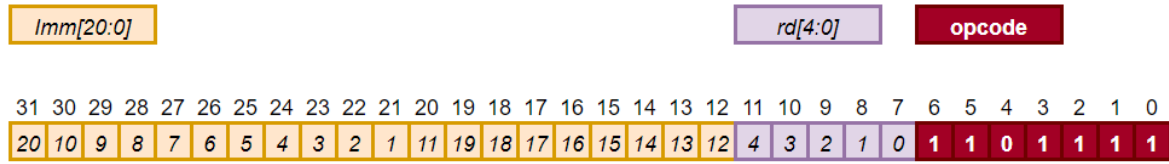


Figure 36. JAL Instruction Bit sequence

JAL (Jump and Link) instruction loads PC+4 to rd and adds PC and sign extended immediate value and loads this value to PC. That makes the program can remember where the jump operation happened and could come back and continue. $Imm[0] = 0$.

$$rd \leq PC + 4, \quad PC \leq PC + sxt(Imm)$$

JALR

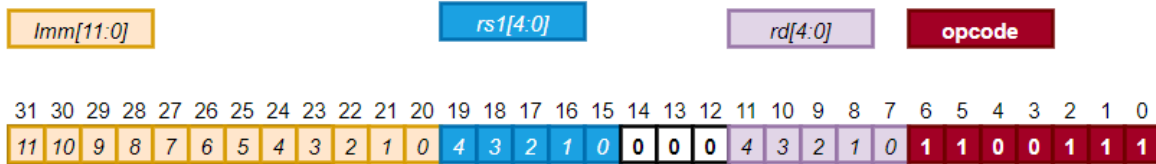


Figure 37. JALR Instruction Bit sequence

JALR (Jump and Link register) instruction loads PC+4 to rd and adds rs1 and sign extended immediate value and loads this value to PC. That makes the program can remember where the jump operation happened and could come back and continue. After the addition, LSB of result is made 0.

$$rd \leq PC + 4, \quad PC \leq \{(rs1 + sxt(Imm))[31:1], \quad 1'b0\}$$



Mnemonic	Name	Description
LUI	Load upper immediate	$rd = imm \ll 12$ (fills lower 12 bits with zeros)
AUIPC	Add upper immediate to pc	$pc = pc + (imm \ll 12)$ (fills lower 12 bits of imm with zeros)
JAL	Jump and link	$rd = pc + 4$, $pc = pc + sxt(imm)$ (LSB of immediate is zero)
JALR	Jump and link register	$rd = pc + 4$, $pc = rs1 + sxt(imm)$ (after the addition LSB of result is made 0)
BEQ	Branch equal	If $rs1 == rs2$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
BNE	Branch not equal	If $rs1 != rs2$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
BLT	Branch less than signed	If $signed(rs1) < signed(rs2)$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
BGE	Branch greater than signed	If $signed(rs1) >= signed(rs2)$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
BLTU	Branch less than unsigned	If $unsigned(rs1) < unsigned(rs2)$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
BGEU	Branch greater than or equal unsigned	If $unsigned(rs1) >= unsigned(rs2)$ then $pc = pc + sxt(imm)$ else $pc = pc + 4$ (LSB of immediate is zero)
LB	Load signed byte	Effective address = $rs1 + sxt(imm)$ $rd = sxt(M[effective_address][7:0])$
LH	Load signed half-word	Effective address = $rs1 + sxt(imm)$ $rd = sxt(M[effective_address][15:0])$
LW	Load word	Effective address = $rs1 + sxt(imm)$ $rd = M[effective_address][31:0]$
LBU	Load unsigned byte	Effective address = $rs1 + sxt(imm)$ $rd = sxt(M[effective_address][7:0])$
LHU	Load unsigned half-word	Effective address = $rs1 + sxt(imm)$ $rd = sxt(M[effective_address][15:0])$
SB	Store byte	Effective address = $rs1 + sxt(imm)$ $M[effective_address][7:0] = rs2[7:0]$
SH	Store half-word	Effective address = $rs1 + sxt(imm)$ $M[effective_address][15:0] = rs2[15:0]$
SW	Store word	Effective address = $rs1 + sxt(imm)$ $M[effective_address][31:0] = rs2[31:0]$
ADDI	Add immediate	$rd = rs1 + sxt(imm)$
SLTI	Set less than immediate	If $signed(rs1) < signed(sxt(imm))$ then $rd = 1$ else $rd = 0$
SLTIU	Set less than unsigned immediate	If $unsigned(rs1) < unsigned(sxt(imm))$ then $rd = 1$ else $rd = 0$
XORI	Logical XOR with immediate	$rd = rs1 \oplus sxt(imm)$ (zxt: zero extend)
ORI	Logical OR with immediate	$rd = rs1 \mid sxt(imm)$
ANDI	Logical AND with immediate	$rd = rs1 \& sxt(imm)$
SLLI	Logical left shift by immediate	$Rd = rs1 \ll imm$
SRLI	Logical right shift by immediate	$Rd = rs1 \gg imm$
SRAI	Arithmetic right shift by immediate	$Rd = rs1 \ggg imm$ (sign bit of $rs1$ copied into upper bits)
ADD	Addition	$Rd = rs1 + rs2$
SUB	Subtraction	$Rd = rs1 - rs2$
SLL	Logical left shift	$Rd = rs1 \ll rs2[4:0]$
SLT	Set less than	If $signed(rs1) < signed(rs2)$ then $rd = 1$ else $rd = 0$
SLTU	Set less than unsigned	If $unsigned(rs1) < unsigned(rs2)$ then $rd = 1$ else $rd = 0$
XOR	Logical XOR	$Rd = rs1 \oplus rs2$
SRL	Logical right shift	$rd = rs1 \gg rs2[4:0]$
SRA	Arithmetic right shift	$Rd = rs1 \ggg rs2[4:0]$ (sign bit of $rs1$ copied into upper bits)
OR	Logical OR	$Rd = rs1 \mid rs2$
AND	Logical AND	$Rd = rs1 \& rs2$