**RV32I Instruction Set**

**Part A**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Type** | **Name** | **Description** | **Note** |
| LUI | U | Load Upper Immediate | rd <= imm << 12 |  |
| AUIPC | U | Add Upper Imm to PC | rd <= PC + (imm << 12) |  |
| ADD | R | ADD | rd <= rs1 + rs2 |  |
| SUB | R | SUB | rd <= rs1 – rs2 |  |
| AND | R | AND | rd <= rs1 & rs2 |  |
| OR | R | OR | rd <= rs1 | rs2 |  |
| XOR | R | XOR | rd <= rs1 ^ rs2 |  |
| SLL | R | Shift Left Logical | rd <= rs1 << rs2 |  |
| SRL | R | Shift Right Logical | rd <= rs1 >> rs2 |  |
| SRA | R | Shift Right Arithmetic\* | rd <= rs1 >> rs2 | msb extends |
| SLT | R | Set Less Than | rd = (rs1 < rs2)?1:0 |  |
| SLTU | R | Set Less Than (U) | rd = (rs1 < rs2)?1:0 | zero extends |
| ADDI | I | ADD Immediate | rd <= rs1 + imm |  |
| ANDI | I | AND Immediate | rd <= rs1 & imm |  |
| ORI | I | OR Immediate | rd <= rs1 | imm |  |
| XORI | I | XOR Immediate | rd <= rs1 ^ imm |  |
| SLTI | I | Set Less Than Immediate | rd = (rs1 < imm)?1:0 |  |
| SLTIU | I | Set Less Than Immediate (U) | rd = (rs1 < imm)?1:0 | zero extends |
| SLLI | I | Shift Left Logical Imm | rd <= rs1 << imm[0:4] |  |
| SRLI | I | Shift Right Logical Imm | rd <= rs1 >> imm[0:4] |  |
| SRAI | I | Shift Right Arith Imm | rd <= rs1 >> imm[0:4] | msb extends |
| LB | I | Load Byte | rd <= rd = M[rs1+imm][0:7] |  |
| LH | I | Load Half | rd = M[rs1+imm][0:15] |  |
| LW | I | Load Word | rd = M[rs1+imm][0:31] |  |
| LBU | I | Load Byte (U) | rd = M[rs1+imm][0:7] | zero extends |
| LHU | I | Load Half (U) | rd = M[rs1+imm][0:15] | zero extends |
| SB | S | Store Byte | m[rs1+imm][0:7] = rs2[0:7] |  |
| SH | S | Store Half | m[rs1+imm][0:15] = rs2[0:15] |  |
| SW | S | Store Word | m[rs1+imm][0:31] = rs2[0:31] |  |
| BEQ | B | Branch == | if (rs1 == rs2) PC += imm |  |
| BNE | B | Branch != | if (rs1 != rs2) PC += imm |  |
| BLT | B | Branch < | if (rs1 < rs2) PC += imm |  |
| BGE | B | Branch >= | if (rs1 >= rs2) PC += imm |  |
| BLTU | B | Branch < (U) | if (rs1 < rs2) PC += imm | zero extends |
| BGEU | B | Branch >= (U) | if (rs1 >= rs2) PC += imm | zero extends |
| JAL | J | Jump and Link | rd=PC+4; PC += imm |  |
| JALR | I | Jump and Link Reg | rd=PC+4; PC = rs1+imm |  |
| ECALL | I | Environment Call | Transfer control to OS |  |
| EBREAK | I | Environment Break | Transfer control to debugger |  |

**Part B**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | | | |
| LUI x1,0x14 | 00000000000000010100 | 00001 | 0110111 |
| x1 <= 0x14 << 12 | imm[31:12] | rd | opcode |
| U-type |  | | |
| Write immediate value on the most significant 20 bits of the value from the register address 1. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | | | |
| LUI x2,0x9 | 00000000000000001001 | 00010 | 0110111 |
| x2 <= 0x9 << 12 | imm[31:12] | rd | opcode |
| U-type |  | | |
| Write immediate value on the most significant 20 bits of the value from the register address 2. | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | | | | | | | |
| SUB x2,x1,x2; | 0100000 | 00010 | 00001 | 000 | 00010 | 0110011 |
| x2 <= x1-x2 | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| R-type |  | | | | | |
| Subtract value of the register address 2 from the value of the register address 1 and write result on the register address 2. | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | | | | | | |
| SRLI x3,x2,0x2 | 0000000 | 00100 | 00010 | 101 | 00011 | 0010011 |
| x3 <= x2 >> 0x2 | imm[11:5] | imm[4:0] | rs1 | funct3 | rd | opcode |
| I-type |  | | | | | |
| Shift right the value from register address 2 by immediate value and write the result on the value from register address 3. | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | | | | | | |
| SLT x4,x1,x3 | 0000000 | 00011 | 00001 | 101 | 00100 | 0110011 |
| x4 <= (x1< x3)?1:0 | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| R-type |  | | | | | |
| Set value from the register address 4 with 1 if the value from register address 1 is smaller than the value from the register address 3. | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | | | | | | |
| BEQ x3,x2,0x04 | 0000000 | 00010 | 00011 | 000 | 00100 | 1100011 |
| if (x3 == x2) PC += 0x04 | imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| B-type |  | | | | | |
| Branch by offset if the values from register address 2 and 3 are equal. | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | | | |
| AUIPC x5,0x13 | 00000000000000010011 | 00101 | 0010111 |
| rd<=PC+(0x13<<12) | imm[31:12] | rd | opcode |
| U-type |  | | |
| ADD the PC value and the immediate value then write the result on the value from register address 5. | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | | | | | | | |
| ADD x6,x5,x4; | 0000000 | 00100 | 00101 | 000 | 00110 | 0110011 |
| x6 <= x5 + x4 | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| R-type |  | | | | | |
| Add the values from register address 5 and 4, then write the result on the value from register address 6. | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | | | | | | | |
| SLLI x6,x6,0x4 | 0000100 | | 00100 | 00110 | 001 | 00110 | 0010011 |
| x6 <= x6 << 0x4 | imm[11:5] | imm[4:0] | | rs1 | funct3 | rd | opcode |
| I-type |  | | | | | | |
| Shift the value from register address 6 by immediate value and write the result on the value from register address 6. | | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | | | | | | |
| JAL 0x8, x10 | 0 | 0000001000 | 0 | 00000000 | 01010 | 1101111 |
| rd=PC+4;PC+=imm | imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode |
| J-type |  | | | | | |
| Write the next instruction’s address to the register address 10 and jump to the immediate address. | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11 | | | | | | |
| SRAI x6,x6,0x4 | 0100000 | 00100 | 00110 | 101 | 00110 | 0010011 |
| x6 <= x6 >> 0x4 | imm[11:5] | imm[4:0] | rs1 | funct3 | rd | opcode |
| I-type |  | | | | | |
| Shift right the value from register address 6 by immediate value and write the result on the register address 6. | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 12 | | | | | | |
| AND x7,x6,x5; | 0000000 | 00101 | 00110 | 111 | 00111 | 0110011 |
| x7 <= x6 & x5 | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| R-type |  | | | | | |
| AND the values from register address 5 and 4, then write the result on the register address 7. | | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | | | | | |
| SLTI x8,x7,0xFC | 000011111100 | 00111 | 010 | 01000 | 0010011 |
| x8<=(x7<0xFC)?1:0 | imm[11:0] | rs1 | funct3 | rd | opcode |
| I-type |  | | | | |
| Set the value from the register address 8 with 1 if the value from register address 7 is smaller than the immediate value. | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 14 | | | |
| LUI x9,0x37 | 00000000000000110111 | 01001 | 0110111 |
| x9 <= 0x37 << 12 | imm[31:12] | rd | opcode |
| U-type |  | | |
| Write immediate value on the most significant 20 bits of the value from the register address 9. | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | | | | | |
| ANDI x9,x9,0x0F0 | 000011110000 | 01001 | 111 | 01001 | 0010011 |
| x9 <= x9 & 0x0F0 | imm[11:0] | rs1 | funct3 | rd | opcode |
| I-type |  | | | | |
| AND the immediate value and the value from the register address 9. | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 16 | | | | | | |
| SW x7,0x8(x9); | 0000000 | 01001 | 00111 | 010 | 01000 | 0100011 |
| m[x9+0x8][0:31] = x7[0:31] | imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| S-type |  | | | | | |
| Copy data from register address 7 to memory location from register address 9 with an offset of immediate value. | | | | | | |

**Part C**

0 00000000000000010100000010110111

1 00000000000000001001000100110111

2 01000000001000001000000100110011

3 00000000010000010101000110010011

4 00000000001100001101001000110011

5 00000000001000011000001001100011

6 00000000000000010011001010010111

7 00000000010000101000001100110011

8 00001000010000110001001100010011

9 00000001000000000000010101101111

10 01000000010000110101001100010011

11 00000000010100110111001110110011

12 00001111110000111010010000010011

13 00000000000000110111010010110111

14 00001111000001001111010010010011

15 00000000100100111010010000100011

**Part D**

NOP is an instruction which doesn’t change any visible value however acts as a regular instruction and NOP is encoded as ADDI x0, x0, 0.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NOP | | | | | |
| ADDI x0,x0,0 | 000000000000 | 00000 | 000 | 00000 | 0110011 |
| x0 <= x0 + 0 | imm[11:0] | rs1 | funct3 | rd | opcode |
| I-type |  | | | | |