# 1. Processor Specification

The Mini SRC is a 32-bit machine, having a 32-bit datapath and sixteen 32-bit registers R0 to R15, with R0 to R7 as general-purpose registers, R8 and R9 as the return value registers, R10 to R13 as the argument registers, R14 as the stack pointer (SP), and R15 as the return address register (RA), holding the return address for a *jal* instruction. It also has two dedicated 32-bit registers HI and LO for multiplication and division instructions. Note that Mini SRC does not have a condition code register. Rather, it allows any of the general-purpose registers to hold a value to be tested for conditional branching. The memory unit is 512 words.
The following is a formal definition of the Mini SRC.

## *Processor State*

| | |
|---|---|
| PC<31..0>: | 32-bit Program Counter (PC) |
| IR<31..0>: | 32-bit Instruction Register (IR) |
| R[0..15]<31..0>: | Sixteen 32-bit registers named R[0] through R[15] |
| R[0..7]<31..0>: | Eight general-purpose registers |
| R[8..9]<31..0>: | Two Return Value Registers |
| R[10..13]<31..0>: | Four Argument Registers |
| R[14]<31..0>: | Stack Pointer (SP) |
| R[15]<31..0>: | Return Address Register (RA) |
| | |
| HI<31..0>: | 32-bit HI Register dedicated to keep the high-order word of a Multiplication product, or the Remainder of a Division operation |
| LO<31..0>: | 32-bit LO Register dedicated to keep the low-order word of a Multiplication |

product, or the Quotient of a Division operation

### *Memory State*

| | |
|---|---|
| Mem[0..511]<31..0>: | 512 words (32 bits per word) of memory |
| MDR<31..0>: | 32-bit memory data register |
| MAR<31..0>: | 32-bit memory address register |

### *I/O State*

| | |
|---|---|
| In.Port<31..0>: | 32-bit input port |
| Out.Port<31..0>: | 32-bit output port |
| Run.Out: | Run/halt indicator |
| Stop.In: | Stop signal |
| Reset.In: | Reset signal |

The *Arithmetic Logic Unit* (ALU) performs 12 operations: addition, subtraction, multiplication, division, shift right, shift left, rotate right, rotate left, logical AND, logical OR, Negate (2's complement), and NOT (1's complement).

The instructions in Mini SRC are one-word (32-bit) long each. They can be categorized as Load and Store instructions, Arithmetic and Logical instructions, Conditional Branch and Jump instructions, Input/Output instructions, and miscellaneous instructions. There are no push and pop instructions (they can be implemented by other instructions). The following addressing modes are supported: Direct, Indexed, Register, Register Indirect, Immediate, and Relative.

## 1.1 Instruction Formats

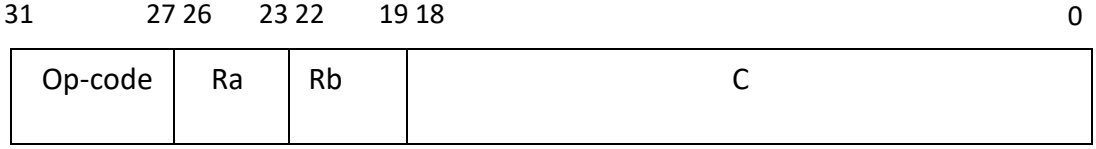
There are five instruction formats, as shown in the table below.

| Name | Fields | | | | | Comments |
|---|---|---|---|---|---|---|
| Field size | 31..27<br>5 bits | 26..23<br>4 bits | 22..19<br>4 bits | 18..15<br>4 bits | 14..0<br>15 bits | All instructions are 32-bit long |
| **R-Format** | OP-code | Ra | Rb | Rc | Unused | Arithmetic/Logical |
| **I-Format** | OP-code | Ra | Rb | Constant C/Unused | | Arithmetic/Logical; Load/Store; Imm. |
| **B-Format** | OP-code | Ra | C2 | Constant C | | Branch |
| **J-Format** | OP-code | Ra | Unused | | | Jump; Input/Output; Special |
| **M-Format** | OP-code | Unused | | | | Misc. |

The instruction formats for different categories are detailed below:

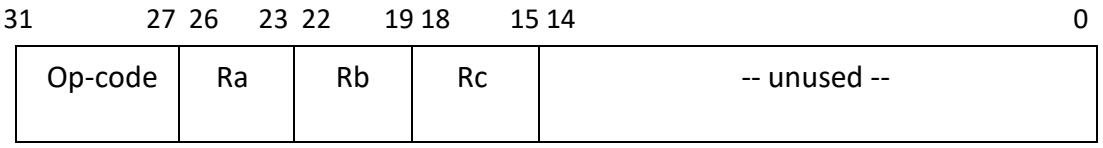
- Load and Store instructions: operands in memory can be accessed only through load/store instructions.

(a) ld, ldi, st I-Format

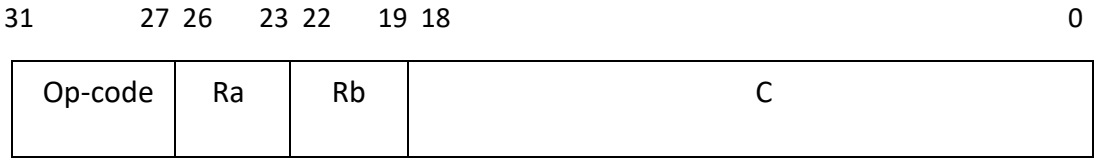


- Arithmetic and Logical instructions:

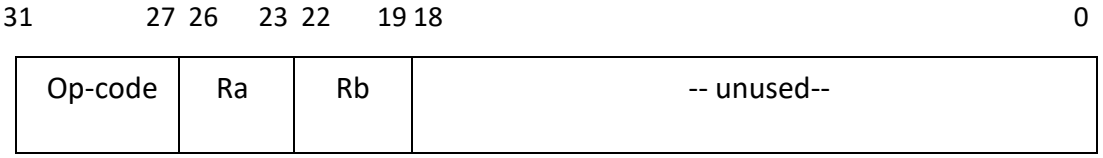
  (a) add, sub, and, or, shr, shl, ror, rol R-Format



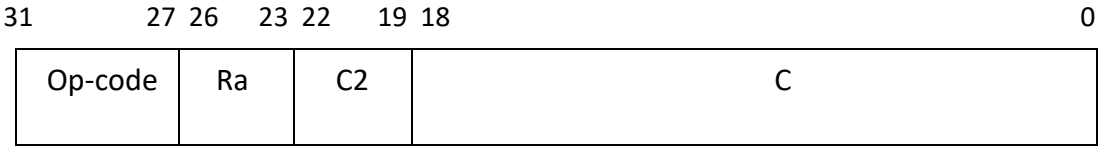

  (b) addi, andi, ori I-Format



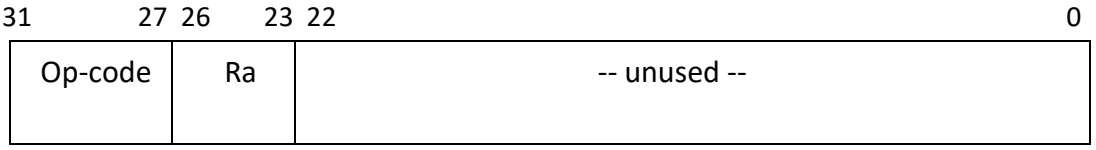

  (c) mul, div, neg, not I-Format



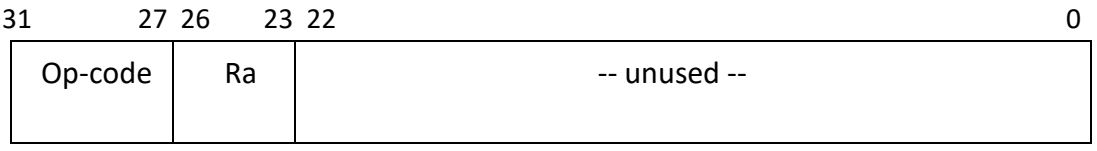

- Branch instructions: brzr, brnz, brmi, brpl B-Format



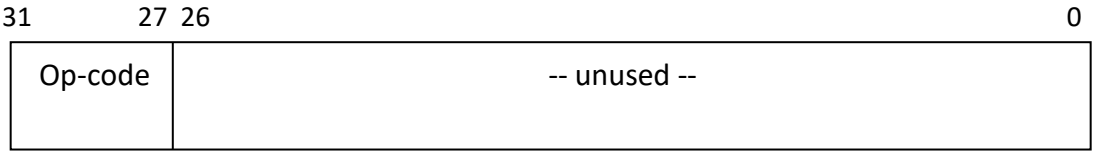

- Jump instructions: jr, jal J-Format




- Input/Output and MFHI/MFLO instructions: in, out, mfhi, mflo J-Format



- Miscellaneous instructions: nop, halt M-Format




Op-code: specifies the operation to be performed.
Ra, Rb, Rc: 0000: R0, 0001: R1, …, 1111: R15
C: constant (data or address)
C2: condition --00: branch if zero
--01: branch if nonzero
--10: branch if positive
--11: branch if negative

Notation: x: 0 or 1
- : unused

## 1.2 Instructions

The instructions (with their op-code patterns shown in parentheses) perform the following operations:

### Load and Store Instructions

**ld, ldi, st:**

| | | Assembly language | |
|---|---|---|---|
| ld: Load Direct<br>(00000xxxx0000xxxxxxxxxxxxxxxxxxxx) | R[Ra] ← M[C (sign-extended) ]<br>Direct addressing, Rb = R0 | ld | Ra, C |
| ld: Load Indexed/Register Indirect<br>(00000xxxxxxxxxxxxxxxxxxxxxxxxxxx) | R[Ra] ← M[R[Rb] + C (sign-extended)]<br>Indexed addressing, Rb ≠ R0<br>If C = 0 → Register Indirect addressing | ld | Ra, C(Rb) |
| ldi: Load Immediate<br>(00001xxxx0000xxxxxxxxxxxxxxxxxxxx) | R[Ra] ← C (sign-extended)<br>Immediate addressing, Rb = R0 | ldi | Ra, C |
| (00001xxxxxxxxxxxxxxxxxxxxxxxxxxx) | R[Ra] ← R[Rb] + C (sign-extended)<br>Immediate addressing, Rb ≠ R0<br>If C = 0 → instruction acts like a simple register transfer<br>If C ≠ 0 and Ra = Rb → Increment/decrement instruction | ldi | Ra, C(Rb) |
| st: Store Direct<br>(00010xxxx0000xxxxxxxxxxxxxxxxxxxx) | M[C (sign-extended)] ← R[Ra]<br>Direct addressing, Rb = R0 | st | C, Ra |
| st: Store Indexed/Register Indirect<br>(00010xxxxxxxxxxxxxxxxxxxxxxxxxxx) | M[R[Rb] + C (sign-extended)] ← R[Ra]<br>Indexed addressing, Rb ≠ R0 | st | C(Rb), Ra |

If C = 0 ➔ Register Indirect addressing

## Arithmetic and Logical Instructions

**(a): add, sub, and, or, shr, shl, ror, rol**

| | | | |
|---|---|---|---|
| add: Add<br>(00011xxxxxxxxxxxx----------) | R[Ra] ← R[Rb] + R[Rc] | add | Ra, Rb, Rc |
| sub: Sub<br>(00100xxxxxxxxxxxx----------) | R[Ra] ← R[Rb] - R[Rc] | sub | Ra, Rb, Rc |
| shr: Shift Right<br>(00101xxxxxxxxxxxx----------) | Shift right R[Rb] into R[Ra] by count in R[Rc] | shr | Ra, Rb, Rc |
| shl: Shift Left<br>(00110xxxxxxxxxxxx----------) | Shift left R[Rb] into R[Ra] by count in R[Rc] | shl | Ra, Rb, Rc |
| ror: Rotate Right<br>(00111xxxxxxxxxxxx----------) | Rotate right R[Rb] into R[Ra] by count in R[Rc] | ror | Ra, Rb, Rc |
| rol: Rotate Left<br>(01000xxxxxxxxxxxx----------) | Rotate left R[Rb] into R[Ra] by count in R[Rc] | rol | Ra, Rb, Rc |
| and: AND<br>(01001xxxxxxxxxxxx----------) | R[Ra] ← R[Rb] ∧ R[Rc] | and | Ra, Rb, Rc |
| or: OR<br>(01010xxxxxxxxxxxx----------) | R[Ra] ← R[Rb] ∨ R[Rc] | or | Ra, Rb, Rc |

**(b): addi, andi, ori**

| | | | |
|---|---|---|---|
| addi: Add Immediate<br>(01011xxxxxxxxxxxxxxxxxxxxxxxxxx) | R[Ra] ← R[Rb] + C (sign-extended)<br>Immediate addressing<br>If C = 0 ➔ instruction acts like a simple register transfer<br>If C ≠ 0 and Ra = Rb ➔ Increment/decrement instruction<br>Similar to ldi, however Rb can be any register | addi | Ra, Rb, C |
| andi: AND Immediate<br>(01100xxxxxxxxxxxxxxxxxxxxxxxxxx) | R[Ra] ← R[Rb] ∧ C (sign-extended)<br>Immediate addressing | andi | Ra, Rb, C |
| ori: OR Immediate<br>(01101xxxxxxxxxxxxxxxxxxxxxxxxxx) | R[Ra] ← R[Rb] ∨ C (sign-extended)<br>Immediate addressing | ori | Ra, Rb, C |

**(c): mul, div, neg, not**

| | | |
|---|---|---|
| mul: Multiply<br>(01110xxxxxxxx ---------- ) | HI, LO ← R[Ra] × R[Rb] | mul Ra, Rb |
| div: Divide<br>(01111xxxxxxxx ---------- ) | HI, LO ← R[Ra] ÷ R[Rb] | div Ra, Rb |
| neg: Negate<br>(10000xxxxxxxx ---------- ) | R[Ra] ← - R[Rb] | neg Ra, Rb |
| not: NOT<br>(10001xxxxxxxx ---------- ) | R[Ra] ← R[Rb] | not Ra, Rb |

## Conditional Branch Instructions

brzr, brnz, brmi, brpl

| | | |
|---|---|---|
| Branch<br>(10010xxxx--xxxxxxxxxxxxxxxxxxxx) | PC ← PC + 1 + C (sign-extended) | if R[Ra] meets the condition |
| Condition: | --00: branch if zero | brzr Ra, C |
| | --01: branch if nonzero | brnz Ra, C |
| | --10: branch if positive | brpl Ra, C |
| | --11: branch if negative | brmi Ra, C |

## Jump Instructions

jr, jal

| | | |
|---|---|---|
| jr: return from procedure<br>(10011xxxx ---------- ) | PC ← R[Ra]<br>If Ra = R15, it is for procedure return | jr Ra |
| jal: jump and link<br>(10100xxxx ---------- ) | R[15] ← PC + 1<br>PC ← R[Ra] | jal Ra |

## Input/Output and MFHI/MFLO Instructions

in, out, mfhi, mflo

| | | |
|---|---|---|
| in: Input<br>(10101xxxx.....) | R[Ra] ← In.Port | in Ra |
| out: Output<br>(10110xxxx.....) | Out.Port ← R[Ra] | out Ra |

| | | | |
|---|---|---|---|
| mfhi: Move from HI<br>(10111xxxx.....) | R[Ra] ← HI | mfhi | Ra |
| mflo: Move from LO<br>(11000xxxx.....) | R[Ra] ← LO | mflo | Ra |

## Miscellaneous Instructions

nop, halt

| | | | |
|---|---|---|---|
| nop: No-operation<br>(11001.....) | Do nothing | nop | |
| halt: Halt<br>(11010.....) | Halt the control stepping process | halt | |