RISC Assembler Documentation (`risc_assembler.py`)

Overview

This document explains how to use the Python-based two-pass assembler (`risc_assembler.py`) designed for a 32-bit custom RISC ISA compatible with a Logisim-based CPU.

Usage

Interactive Mode

```
python risc_assembler.py
```

- Enter assembly instructions line-by-line.
- Submit a blank line to finish and see the generated hex.

Batch Mode

```
python risc_assembler.py input.asm -o output.hex --sym symbols.txt
```

- `input.asm`: Your assembly source file
- `-o output.hex`: Output file for the machine code (hex)
- `--sym symbols.txt`: Optional file to save label-symbol mappings

Instruction Formats

R-Type (Opcode = 0)

```
MNEMONIC Rd, Rs1, Rs2
```

Example:

```
ADD R1, R2, R3
```

I-Type (Opcodes 1–16)

```
MNEMONIC Rd, Rs1, immediate
```

Examples:

```
ADDI R1, R2, 10
SET R3, 0x1234
```

Memory Instructions

`LW` (Load Word)

```
LW Rd, Rs1, imm    ; Standard format
LW Rd, imm(Rs1)    ; MIPS-style format
```

- `Rd`: Register to load into
- `Rs1`: Base address register
- `imm`: Offset

`SW` (Store Word)

```
SW Rs1, Rs2, imm   ; Standard format
SW Rs2, imm(Rs1)   ; MIPS-style format
```

- `Rs1`: Base address register
- `Rs2`: Register to store from
- `imm`: Offset

Branch Instructions (SB-Type)

```
BEQ R1, R2, label
```

- Automatically calculates PC-relative offset from label

Supported Instructions

R-Type (Opcode 0)

- Arithmetic/Logic: `ADD`, `SUB`, `MUL`, `XOR`, `OR`, `AND`, `NOR`, `SEQ`, `SLT`, `SLTU`
- Shifts: `SLL`, `SRL`, `SRA`, `ROR`

I-Type (Opcodes 1–16)

- Immediate operations: `ADDI`, `ORI`, `ANDI`, `NORI`, `XORI`, `SEQI`, `SLTI`, `SLTIU`
- Shifts with immediate: `SLLI`, `SRLI`, `SRAI`, `RORI`
- Constants: `SET`, `SSET`
- Load/jump: `LW`, `JALR`

SB-Type (Opcodes 17–23)

- Store: `SW`
- Branches: `BEQ`, `BNE`, `BLT`, `BGE`, `BLTU`, `BGEU`

Output Format

```
v2.0 raw
<hex values>
```

- Compatible with Logisim's memory file format

Labels

- Labels must end with a colon and appear on their own line.

```
loop:
  ADD R1, R1, R2
  BNE R1, R3, loop
```

Syntax Notes

- Comments start with `;` or `#`
- Immediate values can be decimal or hexadecimal (e.g. `0x10`)
- Registers must be named `R0` – `R31`
- `R0` is hardwired to zero (constant zero)

Example Program

```
SET R1, 10
SET R2, 20
ADD R3, R1, R2
SET R4, 100
SW R4, R3, 0
LW R5, R4, 0
BEQ R3, R5, equal
SET R6, 99
equal:
SET R6, 42
BEQ R0, R0, -1
```

Assembler developed for Spring 2025 RISC CPU project by Mohamed Wageh with ChatGPT.