Simple Risc Assembler Rules

1. One Instruction or Label per Line:

   - Each line may contain a single instruction or a label.

   - Labels must end with a colon (:) and can appear on a line by themselves or alongside an instruction.

2. Comment Syntax:

   - Any text following a slash (/) or a semicolon (;) on a line is treated as a comment and is ignored by the assembler.

3. Comma Handling:

   - Commas (,) in the input are automatically replaced by spaces.

   - Operands can be separated by commas or spaces.

4. Case-Insensitive Opcodes:

   - The opcode (instruction mnemonic) is converted to lowercase before processing.

5. Opcode Mapping:

   - The assembler uses a predefined mapping to convert mnemonics (e.g., "add", "sub", etc.) into their corresponding binary opcodes.

6. Register Definitions:

   - Valid registers are r1 through r15.

   - Each register is mapped to a unique 4-bit binary string (e.g., "r1" → "0001", "r2" → "0010", etc.).

7. Label Handling:

   - Tokens ending with a colon (:) are stored as labels with their corresponding line number (address).

   - Labels are referenced in branch or call instructions.

8. Instruction Addressing Types:

- Type 0 (No Operands): For instructions like nop, ret, and hlt.

  --> Format: The opcode is padded with zeros to create a 32-bit word.

- Type 1 (Single Operand - Branch Instructions): For instructions like b, beq, bgt, and call.

  --> Operand can be a label or an immediate value (hexadecimal values starting with 0x or 0X are allowed).

  --> For labels, an offset is calculated relative to the current instruction.

- Type 2 (Two Operands): For instructions like mov, cmp, and not.

  --> If the second operand starts with r (or R), it is treated as a register; otherwise, it is an immediate value.

- Type 3 (Three Operands): For arithmetic operations such as add, sub, mul, etc.

  --> If the third operand begins with r (or R), all operands are treated as registers.

  --> Otherwise, the third operand is considered an immediate value, with an associated modifier.

- Type 4 (Memory Access): For load/store instructions (ld and st).

  --> Format: opcode rd, imm[rs1]

  - rd: destination (or source) register.

  - imm[rs1]: an immediate value followed by a base register in square brackets.

9. Immediate Value Formats:

- Numeric Extraction: The assembler extracts the numeric part from operands (supports negative values with a minus sign).

- Hexadecimal Notation: Immediate values can be specified in hexadecimal if prefixed with 0x or 0X.

- Fixed Bit Widths:

  --> Branch offsets (Type 1) are converted to a 27-bit two's complement binary string.

  --> Immediate values in Type 2 and Type 3 are converted to a 16-bit binary string.

--> The immediate value in memory instructions (Type 4) is converted to a 4-bit binary string.

10. Modifier Handling for Variants:

- For instructions with variants (indicated by an extra character such as "h" or "u"), the assembler adjusts the opcode and appends a modifier field (e.g., "01" or "10") accordingly.

11. Error Reporting:

- If an instruction does not have the required number of operands, the assembler outputs an error message.

- If an operand refers to an unknown register or if an immediate operand has no numeric part, an error is reported.

12. Machine Code Word Size:

- Every assembled instruction is converted into a 32-bit binary string.

13. Hexadecimal Output:

- After converting to binary, the machine code is further converted into a hexadecimal representation.

- Each hexadecimal value is padded with leading zeros to a fixed width and is written to "hexfile.hex".