Assembly Instruction Encoder

Overview

This Python script serves as an assembler for a custom instruction set architecture (ISA). It takes an assembly code file containing instructions written in a specific format and converts them into their binary representation based on a predefined instruction mapping.

Usage

The script can be executed from the command line with the following syntax:

```
python assembler.py input_file output_file
```

- `input_file`: Path to the assembly code file containing instructions.
- `output_file`: Path to the output file where the binary representations will be written `default output.txt`.

Instruction Mapping

Instructions are categorized based on the number of operands they take.

- **0 Operand Instructions:**
  - `NOP`: No operation.
  - `RET`: Return from subroutine.
  - `RTI`: Return from interrupt.
- **1 Operand Instructions:**
  - Arithmetic and logical operations such as `NOT`, `NEG`, `INC`, `DEC`, etc.
  - Memory operations like `IN`, `OUT`, `LDM`, `Push`, `POP`, etc.
  - Jump instructions: `JZ`, `JMP`, `CALL`.
- **2 Operand Instructions:**
  - `SWAP`: Swap the contents of two registers.
  - `CMP`: Compare two operands.
  - `ADDI`: Add immediate value to a register.
- **3 Operand Instructions:**
  - Arithmetic and logical operations: `ADD`, `SUB`, `XOR`, `OR`, `AND`.

Input File Format

The assembly code file should adhere to the following rules: - Each instruction is written on a separate line. - Spaces are required between each parameter in an instruction. - Instructions are case-insensitive. - Registers are represented as `R0` to `R7`. - Immediate values can be specified in decimal, hexadecimal (with 'H' suffix), octal (with 'O' suffix), or binary (with 'B' suffix).

Error Handling

The script performs basic error checking to ensure the integrity of the input assembly code. It checks for issues such as unknown instructions, incorrect operand counts, invalid register references, and out-of-range immediate values.

Output

The binary representation of the input assembly code is written to the specified output file. Each line corresponds to an instruction, with immediate values represented in binary when applicable.

Example

Suppose the input file (`input.asm`) contains the following code:

```
sub R1 R2 R3
LDM R4 FFh
JMP R7
```

output

```
1100101001111010
0110000001001001
0000000011111111
0111100000001100
```

Executing the script:

```
python assembler.py input.txt output.txt
```

The output file (`output.txt`) will contain the binary representation of the instructions.

Note: This documentation provides an overview of the script's functionality and usage. For specific details on instruction formats and encoding, refer to the comments within the script (`assembler.py`).