# Register Renaming Simulator

The provided code is a simplified register renaming simulator for a basic instruction set architecture (ISA). It primarily handles output dependencies. Output dependencies refer to a situation in which two instructions write to same register, which can be resolved by renaming the registers. The code simulates the execution of a series of instructions and uses register renaming to eliminate data hazards and improve instruction issue efficiency. Below is an overview of the approach:

**Data Structures**: The code defines two data structures: Register and Instruction(defined using structs). Register structure represents a physical register with value and valid fields. The valid field is used to indicate whether a physical register is currently in use or available. Instruction structure stores information about an instruction, such as the operation (op), source registers (src1 and src2), destination register (dest), a flag to track output renaming (outputRenamed), memory addressing mode (memAddressMode), and memory address (memAddress).

The code then defines an array physicalRegisters to represent a pool of physical registers. These registers are used for renaming and breaking dependencies between instructions. The allocatePhysicalRegister function is responsible for allocating a physical register from the pool. It keeps track of the next available physical register. If there are no available physical registers, it returns -1. The allocatePhysicalRegisters function takes an Instruction as a parameter and performs register renaming. It checks whether the source registers are valid and if not, it renames them to physical registers. If the destination register is not -1 (indicating no write-back), it allocates a physical register for the destination and marks it as valid. The issue function simulates the issue stage of the processor pipeline and prints the renamed instruction. It takes an Instruction as input and prints the details of the instruction, including the renamed registers if necessary.

**Main Function**: The main function initializes the physical registers and the mapping from architectural to physical registers. It then reads the number of instructions and processes each instruction sequentially. It prompts the user to enter instructions and parses the instruction format. For each instruction, it determines the operation type, source registers, destination register, and other attributes and creates an Instruction structure.
It then calls the issue function to simulate the instruction issue, including register renaming if necessary. The code will allocate physical registers, rename registers, and print out the renamed instructions.

**Input Processing:** The code reads input instructions from a file ("input.txt") and processes them. Instructions can be of two types:

1. Register-to-Register (e.g., "ADD R1, R2, R3")
2. Memory Load/Store (e.g., "LD R1, 100(R2)")

The code performs error checking and reports unknown instructions or invalid instruction formats.

**Sample Input:**

```
7
ADD R1, R2, R3
SUB R4, R1, R5
LD R6, 100(R7)
SD R1, 200(R2)
MUL R8, R1, R9
ADD R10, R11, R12
INVALID R13, R14, R15
```

**Sample Output:**

```
 ADD R0, R0, R0
SUB R1, R0, R4
LD R2, 100(R7)
SD R0, 200(R0)
MUL R3, R0, R8
ADD R4, R5, R6
Unknown instruction: INVALID
```