What is Peephole Optimization?

Peephole Optimization is a technique used in compiler design to make small improvements in a **tiny section of code** (called a **peephole** or **window**).
It replaces a part of code with a **shorter and faster version** without changing the output.

It's a **machine-dependent optimization** that helps improve efficiency at the instruction level.

Objectives of Peephole Optimization

- ✅ Improve performance
- ✅ Reduce memory usage
- ✅ Decrease code size
- ✅ Eliminate unnecessary instructions

Peephole Optimization Techniques:

A. Redundant load and store elimination
B. Constant folding
C. Strength Reduction
D. Null sequences/ Simplify Algebraic Expressions
E. Combine operations
F. Deadcode Elimination

A. Redundant Load and Store Elimination

1. Removes variables that just **copy values** from others.
2. Reduces the number of **unnecessary assignments**.
3. Makes the code **shorter and more efficient**.

B. Constant Folding

1. Performs **arithmetic with constants** during compilation.
2. Reduces **runtime calculations**.
3. Speeds up execution by using **pre-computed values**.

C. Strength Reduction

1. Replaces **expensive operations** (like * and /) with **cheaper ones** (like +, <<, >>).
2. Improves performance by reducing **CPU workload**.
3. Commonly applied in **loops or repeated operations**.

D. Simplify Algebraic Expressions / Null Operations

1. Removes **operations that don't change** the value (like +0, *1).
2. Keeps the code **clean and minimal**.
3. Avoids **unnecessary instructions** in the final code.

E. Combine Operations

1. Combines **multiple steps into one** operation.
2. Reduces the number of instructions.
3. Makes the code **simpler and faster to execute**.

F. Dead Code Elimination

1. Deletes code that is **never used** or **never reached**.
2. Reduces **memory usage** and **code size**.
3. Improves speed by **avoiding useless operations**.

Peephole Optimization Techniques Examples

Technique	Original Code	Optimized Code	What Changed
Redundant Load and Store Elimination	x = 10 y = x z = y result = z + 5	x = 10 result = x + 5	Removed extra copies of the same value
Constant Folding	a = 2 * 5 b = 6 + 4	a = 10 b = 10	Pre-calculated constant expressions
Strength Reduction	total = value * 2 half = value / 2	total = value + value half = value >> 1	Used faster operations instead of * and /
Simplify Algebraic Expressions	score = score + 0 marks = marks * 1 age = age - 0	(All lines removed)	Removed operations that don't change anything
Combine Operations	a = 3 + 2 b = a + 1	b = 3 + 2 + 1 or b = 6	Combined into a single simple calculation
Dead Code Elimination	a = 5 a = 10 return a = a + 1	a = 5 return	Removed code after return that never runs