CSA1455

Compiler Design For Lexical Analysis

Assignment - 2

Name: Sai Lokesh Nalabothu

Regd: 192365023

Branch: CSE - cyber security

Date: 21 - Feburary - 2025

Peehole optimization techniques

Peehole optimization is a Local optimization technique that analyzes a small sequence of instructions in generated code to detect and Eliminate inefficiences. It is applied after Code generation to improve Runtime Performance and Reduce Memory usage.

PARAMETERS

1. Local optimization techniques

Local optimization operate within a limited scope, such as a single basic block, to improve efficiency. Some common techniques include:

- Redundant Instruction Elimination

- Constant folding

- Strength Reduction

- Algebraic Simplification

2. Common Subexpression Elimination

CSE identifies and Eliminates duplicate expressions that Compute the same value multiple times within a local scope.

Example:

Before optimization:

```
int a = (x + y) * z;

int b = (x + y) * w;
```

After optimization:

```
int temp = x + y;

int a = temp * z;

int b = temp * w;
```

3. Dead Code Elimination

Dead Code Refers to instructions that do not affect the Programs output. These Can be safely Removed to Reduce Execution overhead.

Example:

Before optimization:

```
int x = 5;
x = 10;
Printf ("%d", x);
```

After optimization:

```
int x = 10;
printf ("%d", x);
```

The first assignment (x = 5) is Removed because x is immediately Reassigned before being used.

1. What is Peehole optimization, and why is it used?

→ Peehole optimization is a local optimization technique that improves the efficiency of Generated code by analyzing and optimizing small segments of Instructions at a time. This technique is applied during the compilation phase, typically after code generation.

WHY?

Peehole optimization is used for several Reasons:

1. Reducing Code size

2. Improving Enecution speed

3. optimizing Register usage

4. Eliminating Redundant Computations

5. Enhancing Performance without Affecting Program Logic.

Example:

Before:

```
MOV R1, A
MOV A, R1
```

After:

```
MOV R1, A
```

2. Explain Redundant instruction Elimination with an Example.

* Redundant instruction elimination is a Peephole optimization technique that Removes unnecessary instructions that do not Contribute to the final output

Example:

Before optimization:

```
int x = 5;
x = 10;
Redundant
Printf("%d", x);
```

After optimization:

```
int x = 10;
Printf("%d", x);
```

Benefits:

- It Eliminates unnecessary operations.
- Fewer instructions Mean Faster Execution.
- Removes Unnecessary loads and Stores.

3. How does Constant folding improve Code efficiency?

* Constant folding is a Compile-time optimization technique that Evaluates Constant Expressions before Execution and Replaces them with their Computed values.

How it improves Code efficiency?

1. Reduces Computation at Run time
2. Decreases Execution time
3. optimizes Memory usage
4. Improves Readability and Maintainability.

Example:

Before :

```
for (inti=0;i<100;i+ +) {

}
```

After:

```
int y = 6;
for (int i=0;i<100;i++)
{

}
```

4. what is dead Code Elimination, and how does it work?

* Dead Code Elimination is a Compiler optimization technique that Removes Code that does not affect the Program's output. This helps Reduce Program's size, improve Execution speed, and optimize Memory usage.

How?

It follows three main steps:

1. Code Analysis
2. Marking unused Code
3. Eliminating Dead Code

Types of Dead Code:

1. Un Reachable Code
2. unused variables
3. Redundant Assignments

5. Compare Local and Global optimizations

Feature	Local optimization	Global optimization
Scope	Small Region	Entire Function or Program
Complexity	simpler, Requirs Minimal analysis	More Complex, Requirs data flow analysis
Code size Reduction	small Reductions	Can significtantly Reduce Program size.
Time & Resources	Fast and Light Weight	Requirs More time and Memory,

* Local optimization is simpler but faster

* Global optimization is More Powerful and improvs overall Performance.