📘 Compiler Design – UNIT V

🔥 Code Optimization

🎯 2 Days Before Exam Notes (RGPV Exam Oriented)

🌟 UNIT V SYLLABUS

📌 Introduction to Code Optimization

- Sources of Optimization
- Optimization of Basic Blocks
- Loops in Flow Graphs
- Dead Code Elimination
- Loop Optimization

📌 Data Flow Analysis

- Global Data Flow Analysis
- Code Improving Transformations
- Data Flow Analysis of Flow Graph

📌 Symbolic Debugging of Optimized Code

🔥 MOST IMPORTANT TOPICS

Topic	Importance
Code Optimization	⭐⭐⭐⭐⭐
Dead Code Elimination	⭐⭐⭐⭐⭐
Loop Optimization	⭐⭐⭐⭐⭐
Basic Block Optimization	⭐⭐⭐⭐
Flow Graph	⭐⭐⭐⭐
Data Flow Analysis	⭐⭐⭐⭐
Code Improving Transformations	⭐⭐⭐⭐
Symbolic Debugging	⭐⭐⭐

🔥 1. INTRODUCTION TO CODE OPTIMIZATION

📌 Definition

Code Optimization is the process of improving intermediate code to:

```
Increase Speed
Reduce Memory Usage
Improve Efficiency
```

without changing program output.

🌟 Objectives of Optimization

✅ Faster execution

✅ Less memory usage

✅ Better CPU utilization

✅ Reduced redundant code

🌟 Example

Before Optimization:

```
a = b + 0;
```

After Optimization:

```
a = b;
```

🌟 Types of Optimization

Type	Description
Machine Independent	Optimization before machine code
Machine Dependent	Optimization after machine code

🌟 Advantages

✅ Improves performance

✅ Reduces execution time

✅ Reduces code size

📌 Conclusion

Code Optimization improves program efficiency without affecting correctness.

🔥 2. SOURCES OF OPTIMIZATION

📌 Definition

Sources of Optimization are program areas where unnecessary operations can be reduced.

🌟 Main Sources

Source	Optimization
Redundant Expressions	Remove repeated computations
Dead Code	Remove unused statements
Constant Expressions	Evaluate at compile time
Loops	Reduce repeated calculations

🌟 Example

```
x = a+b;
y = a+b;
```

Optimization:

```
t1 = a+b;
x = t1;
y = t1;
```

📌 Conclusion

Optimization sources help compiler generate efficient code.

🔥 3. OPTIMIZATION OF BASIC BLOCKS

📌 Definition

Optimization performed inside a basic block is called Basic Block Optimization.

🌟 Basic Block

A sequence of instructions with:

```
Single Entry
Single Exit
```

🌟 Techniques Used

Technique	Purpose
Common Subexpression Elimination	Removes repeated expressions

Technique	Purpose
Constant Folding	Evaluates constants
Copy Propagation	Replaces copied variables

🌟 Example

Before:

```
a = b+c;
d = b+c;
```

After:

```
t1 = b+c;
a = t1;
d = t1;
```

🌟 Advantages

✅ Faster execution
✅ Smaller code size

📌 Conclusion

Basic Block Optimization removes unnecessary operations inside blocks.

🔥 4. LOOPS IN FLOW GRAPHS

📌 Definition

A Loop is a sequence of instructions executed repeatedly.

Flow Graph represents control flow between blocks.

🌟 Example of Loop

```
for(i=0;i<10;i++)
```

🌟 Flow Graph

```
B1 → B2 → B3
      ↑    ↓
     ←----
```

🌟 Components

Component	Meaning
Nodes	Basic Blocks
Edges	Flow of Control

🌟 Advantages

✅ Detects loops

✅ Supports optimization

📌 Conclusion

Flow Graph helps compiler analyze loops and optimize execution.

🔥 5. DEAD CODE ELIMINATION 🔥

VERY IMPORTANT

📌 Definition

Dead Code is code that never affects program output.

Dead Code Elimination removes such useless code.

🌟 Example

```
a = 5;
a = 10;
```

First statement becomes useless.

🌟 Causes of Dead Code

✅ Unused variables

✅ Unreachable statements

✅ Overwritten values

🌟 Advantages

✅ Reduces code size

✅ Faster execution

✅ Better memory usage

🌟 Disadvantages

❌ Complex analysis required

📌 Conclusion

Dead Code Elimination improves efficiency by removing unnecessary statements.

🔥 6. LOOP OPTIMIZATION 🔥 VERY IMPORTANT

📌 Definition

Loop Optimization improves execution efficiency of loops.

🌟 Need

Loops execute repeatedly, so optimization greatly improves performance.

🌟 Techniques

Technique	Purpose
Loop Unrolling	Reduce loop overhead
Code Motion	Move invariant code outside loop
Strength Reduction	Replace expensive operations

🌟 Example

Before:

```
for(i=0;i<n;i++)
{
   x = y*2;
}
```

After:

```
t = y*2;

for(i=0;i<n;i++)
{
   x = t;
}
```

🌟 Advantages

✅ Faster loops
✅ Reduced computations
✅ Better performance

📌 Conclusion

Loop Optimization reduces repeated computations and improves execution speed.

🔥 7. GLOBAL DATA FLOW ANALYSIS

📌 Definition

Global Data Flow Analysis studies flow of data throughout the program.

🌟 Purpose

✅ Detect unused variables
✅ Find optimization opportunities
✅ Analyze variable values

🌟 Example

Checks:

```
Where variable is defined
Where variable is used
```

🌟 Advantages

✅ Better optimization

✅ Improves code quality

📌 Conclusion

Global Data Flow Analysis helps compiler optimize entire program efficiently.

🔥 8. CODE IMPROVING TRANSFORMATIONS

📌 Definition

Transformations that improve code efficiency are called Code Improving Transformations.

🌟 Techniques

Technique	Purpose
Constant Folding	Compute constants early
Copy Propagation	Replace copied variables
Common Subexpression Elimination	Remove repeated expressions
Dead Code Elimination	Remove useless code

🌟 Example

Before:

```
x = 5*10;
```

After:

```
x = 50;
```

🌟 Advantages

✅ Faster execution
✅ Better optimization

📌 Conclusion

Code Improving Transformations generate efficient and optimized programs.

🔥 9. DATA FLOW ANALYSIS OF FLOW GRAPH

📌 Definition

Data Flow Analysis studies movement of data in a flow graph.

🌟 Purpose

✅ Optimization
✅ Error detection
✅ Variable analysis

🌟 Flow Graph

```
B1 → B2 → B3
```

🌟 Applications

✅ Dead code detection
✅ Loop optimization
✅ Register allocation

📌 Conclusion

Data Flow Analysis helps compiler analyze and optimize programs systematically.

🔥 10. SYMBOLIC DEBUGGING OF OPTIMIZED CODE

📌 Definition

Symbolic Debugging helps programmers debug optimized code using source level information.

🌟 Need

Optimization changes:

```
Variable locations
Instruction order
```

making debugging difficult.

🌟 Functions

✅ Maps machine code to source code
✅ Tracks variables
✅ Detects runtime errors

🌟 Advantages

✅ Easier debugging
✅ Better error tracking

🌟 Disadvantages

❌ Complex implementation

📌 Conclusion

Symbolic Debugging helps understand optimized programs during debugging.

🔥 MOST IMPORTANT 14-MARK QUESTIONS

⭐ Very Very Important

1. Explain Code Optimization.
2. Explain Sources of Optimization.
3. Explain Optimization of Basic Blocks.
4. Explain Dead Code Elimination.
5. Explain Loop Optimization.
6. Explain Global Data Flow Analysis.
7. Explain Code Improving Transformations.
8. Explain Data Flow Analysis of Flow Graph.
9. Explain Symbolic Debugging of Optimized Code.

⚡ SMART 2-DAY STUDY PLAN

🌟 DAY 1

MUST STUDY FIRST

✅ Code Optimization

✅ Dead Code Elimination

✅ Loop Optimization

✅ Basic Block Optimization

THEN STUDY

✅ Sources of Optimization

✅ Flow Graph

✅ Code Improving Transformations

🌟 DAY 2

MUST STUDY FIRST

✅ Data Flow Analysis

✅ Global Data Flow Analysis

✅ Symbolic Debugging

LAST REVISION

✅ Constant Folding

✅ Copy Propagation

✅ Common Subexpression Elimination

⚡ ONE NIGHT REVISION

✔ Optimization improves speed and efficiency

✔ Dead code never affects output

✔ Loop optimization reduces repeated work

✔ Basic blocks have single entry and exit

✔ Flow graph shows control flow

✔ Data flow analysis tracks variable usage

✔ Constant folding computes constants early

✔ Copy propagation removes unnecessary copies

✔ Symbolic debugging maps optimized code to source code