📘 Compiler Design – UNIT V

🔥 MOST IMPORTANT 14-MARK QUESTIONS

🎯 Simple + Detailed + RGPV Exam Writing Style

🔥 Q1. Explain Code Optimization.

✍️ Answer:

📌 Definition

Code Optimization is the process of improving intermediate code so that program:

```
Executes Faster
Uses Less Memory
Works Efficiently
```

without changing program output.

🌟 Need of Code Optimization

✅ Improves execution speed

✅ Reduces memory usage

✅ Removes unnecessary instructions

✅ Improves CPU performance

🌟 Example

Before Optimization:

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

After Optimization:

```
a = b;
```

🌟 Types of Optimization

Type	Description
Machine Independent	Before machine code generation
Machine Dependent	After machine code generation

🌟 Advantages

✅ Faster programs

✅ Smaller code size

✅ Better efficiency

🌟 Disadvantages

❌ Compilation time increases
❌ Complex implementation

📌 Conclusion

Code Optimization improves overall program performance without changing program result.

🔥 Q2. Explain Sources of Optimization.

✍️ Answer:

📌 Definition

Sources of Optimization are places in a program where unnecessary operations can be removed or improved.

🌟 Main Sources of Optimization

Source	Optimization
Repeated Expressions	Remove repeated calculations
Dead Code	Remove unused statements
Constant Expressions	Evaluate constants at compile time
Loops	Reduce repeated work

🌟 Example

Before:

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

After:

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

🌟 Types of Optimization Sources

1. Common Subexpression

Repeated expressions are removed.

---

2. Constant Folding

Constant expressions calculated early.

Example:

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

Optimized:

```
x = 50;
```

---

3. Dead Code

Unused code removed.

---

4. Loop Optimization

Repeated calculations inside loops reduced.

🌟 Advantages

✅ Faster execution
✅ Reduced computations
✅ Better memory usage

📌 Conclusion

Optimization sources help compiler generate efficient and optimized code.

🔥 Q3. Explain Optimization of Basic Blocks.

✍️ Answer:

📌 Definition

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

🌟 Basic Block

A sequence of instructions having:

```
Single Entry
Single Exit
```

🌟 Example of Basic Block

```
a = b+c
d = b+c
e = d+f
```

🌟 Optimization Techniques

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

🌟 Example

Before Optimization:

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

After Optimization:

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

🌟 Advantages

✅ Faster execution
✅ Reduced instructions
✅ Better optimization

🌟 Disadvantages

❌ Works only inside one block

📌 Conclusion

Basic Block Optimization improves efficiency by removing unnecessary computations inside blocks.

🔥 Q4. Explain Dead Code Elimination.

✍️ Answer:

📌 Definition

Dead Code is code that never affects program output.

Dead Code Elimination removes such useless code.

🌟 Example

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

Statement:

```
a = 5;
```

is useless because value changes later.

🌟 Causes of Dead Code

✅ Unused variables

✅ Unreachable statements

✅ Overwritten values

🌟 Types of Dead Code

Type	Example
Unreachable Code	Code after return statement
Unused Variables	Variable never used

🌟 Advantages

✅ Reduces code size

✅ Faster execution

✅ Better memory usage

🌟 Disadvantages

❌ Complex analysis needed

📌 Conclusion

Dead Code Elimination improves efficiency by removing unnecessary statements.

🔥 Q5. Explain Loop Optimization.

✍️ Answer:

📌 Definition

Loop Optimization improves efficiency of loops by reducing repeated calculations.

🌟 Need of Loop Optimization

Loops execute many times, so optimization greatly improves performance.

🌟 Example

Before Optimization:

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

After Optimization:

```
t = y*2;

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

🌟 Loop Optimization Techniques

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

🌟 Advantages

✅ Faster loops

✅ Reduced computations

✅ Better CPU performance

🌟 Disadvantages

❌ Increased code complexity

📌 Conclusion

Loop Optimization improves execution speed by minimizing repeated work inside loops.

🔥 Q6. Explain Global Data Flow Analysis.

✍️ Answer:

📌 Definition

Global Data Flow Analysis studies movement of data throughout the entire program.

🌟 Purpose

✅ Detect optimization opportunities

✅ Analyze variable usage

✅ Detect dead code

🌟 Working

Compiler checks:

```
Where variables are defined
Where variables are used
```

🌟 Applications

Application	Purpose
Dead Code Detection	Remove useless statements
Constant Propagation	Replace variables with constants
Register Allocation	Efficient register usage

🌟 Advantages

✅ Better optimization

✅ Efficient execution

✅ Improved code quality

🌟 Disadvantages

❌ Complex analysis

📌 Conclusion

Global Data Flow Analysis helps compiler optimize programs efficiently.

🔥 Q7. Explain Code Improving Transformations.

✍️ Answer:

📌 Definition

Code Improving Transformations are techniques used to improve efficiency of code.

🌟 Main Transformations

Transformation	Purpose
Constant Folding	Compute constants early
Copy Propagation	Remove unnecessary copies
Dead Code Elimination	Remove useless code
Common Subexpression Elimination	Remove repeated expressions

🌟 Example

Before:

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

After:

```
x = 50;
```

🌟 Advantages

✅ Faster execution
✅ Reduced code size
✅ Better optimization

🌟 Disadvantages

❌ Compilation becomes complex

📌 Conclusion

Code Improving Transformations generate efficient and optimized programs.

🔥 Q8. Explain Data Flow Analysis of Flow Graph.

✍️ Answer:

📌 Definition

Data Flow Analysis studies movement of data in a Flow Graph.

🌟 Flow Graph

A graphical representation of control flow between basic blocks.

🌟 Diagram

```
B1 → B2 → B3
```

🌟 Components

Component	Meaning
Nodes	Basic Blocks
Edges	Control Flow

🌟 Purpose of Data Flow Analysis

✅ Detect dead code

✅ Optimize loops

✅ Register allocation

🌟 Applications

✅ Optimization

✅ Variable analysis

✅ Program analysis

🌟 Advantages

✅ Efficient optimization

✅ Better program analysis

🌟 Disadvantages

❌ Complex implementation

📌 Conclusion

Data Flow Analysis helps compiler improve program efficiency using flow graph analysis.

🔥 Q9. Explain Symbolic Debugging of Optimized Code.

✍️ Answer:

📌 Definition

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

🌟 Need of Symbolic Debugging

Optimization changes:

```
Instruction order
Variable locations
```

which makes debugging difficult.

🌟 Functions

✅ Maps machine code to source code
✅ Tracks variable values
✅ Helps detect runtime errors

🌟 Example

Optimized code may remove:

```
Unused variables
```

Debugger helps trace original source program.

🌟 Advantages

✅ Easier debugging
✅ Better error tracking
✅ Helps understand optimized code

🌟 Disadvantages

❌ Difficult implementation
❌ More debugging complexity

📌 Conclusion

Symbolic Debugging helps programmers understand and debug optimized programs effectively.

⚡ ONE NIGHT REVISION

```
✔ Optimization improves speed and memory usage
✔ Dead code never affects output
✔ Basic blocks have single entry and exit
```

✔ Loop optimization reduces repeated work

✔ Data flow analysis tracks variable usage

✔ Constant folding computes constants early

✔ Copy propagation removes unnecessary copies

✔ Symbolic debugging maps optimized code to source code