📘 Compiler Design – UNIT IV

🔥 MOST IMPORTANT 14-MARK QUESTIONS

🎯 Detailed RGPV Exam Answers (Easy + Long Writing Style)

🔥 Q1. Explain Intermediate Code Generation.

✍️ Answer:

📌 Introduction

Intermediate Code Generation is a compiler phase in which:

```
Machine Independent Code
```

is generated from source program.

It acts as a bridge between:

```
Source Code
        &
Machine Code
```

🌟 Need of Intermediate Code

✅ Simplifies code generation

✅ Supports optimization

✅ Makes compiler machine independent

🌟 Example

Source Program:

```
a = b + c
```

Intermediate Code:

```
t1 = b + c
a = t1
```

🌟 Types of Intermediate Code

Type	Example
Three Address Code	t1 = a+b
Quadruples	(+,a,b,t1)

Type	Example
Triples	(0)+ab

🌟 Advantages

✅ Easy optimization

✅ Platform independent

✅ Simplifies translation

🌟 Disadvantages

❌ Additional compilation phase

❌ Extra memory required

📌 Conclusion

Intermediate Code Generation simplifies compiler design and supports efficient code optimization.

🔥 Q2. Explain Backpatching with Example.

✍️ Answer:

📌 Definition

Backpatching is a technique used to fill incomplete jump addresses later during code generation.

🌟 Need of Backpatching

During compilation:

```
Target labels may not be known immediately
```

So temporary blanks are left.

🌟 Example

Intermediate Code:

```
if a>b goto ___
goto ___
```

Later filled as:

```
if a>b goto L1
goto L2
```

🌟 Working of Backpatching

1. Generate incomplete jump statements
2. Store jump positions in list
3. Fill correct labels later

🌟 Advantages

✅ Simplifies code generation
✅ Useful for loops and conditionals
✅ Supports forward jumps

🌟 Applications

✅ If statements
✅ Loops
✅ Boolean expressions

📌 Conclusion

Backpatching is an important technique used for handling incomplete branch targets during intermediate code generation.

🔥 Q3. Explain Basic Block and Flow Graph.

✍️ Answer:

🌟 Basic Block

📌 Definition

A Basic Block is a sequence of statements having:

```
Single Entry
Single Exit
```

Control enters at beginning and exits at end without interruption.

🌟 Example

```
a = b + c
d = a - e
f = d * g
```

🌟 Characteristics

✅ No branching inside block

✅ Statements execute sequentially

🌟 Advantages

✅ Simplifies optimization

✅ Easy flow analysis

🌟 Flow Graph

📌 Definition

Flow Graph is a graphical representation showing flow of control between basic blocks.

🌟 Diagram

```
B1 → B2 → B3
```

🌟 Components

Component	Meaning
Nodes	Basic Blocks
Edges	Control Flow

🌟 Uses

✅ Program optimization
✅ Loop detection
✅ Program analysis

📌 Conclusion

Basic Blocks and Flow Graphs are used for program optimization and control flow analysis.

🔥 Q4. Explain DAG Representation of Basic Block.

✍️ Answer:

📌 Definition

DAG stands for:

```
Directed Acyclic Graph
```

It is used to represent expressions inside a basic block.

🌟 Purpose of DAG

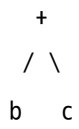
✅ Detect common subexpressions
✅ Eliminate redundant computations
✅ Support optimization

🌟 Example

Statements:

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

🌟 DAG Representation



Both statements use same node.

🌟 Advantages

✅ Removes repeated expressions

✅ Reduces unnecessary computation

✅ Improves efficiency

🌟 Applications

✅ Code optimization

✅ Common subexpression elimination

📌 Conclusion

DAG representation helps generate optimized and efficient code from basic blocks.

🔥 Q5. Explain Peephole Optimization.

✍️ Answer:

📌 Definition

Peephole Optimization is a local optimization technique that examines small sets of instructions and improves them.

🌟 Working

Compiler looks through a:

```
Small Window (Peephole)
```

of instructions and replaces inefficient code with better code.

🌟 Example

Before Optimization:

```
a = a + 0
```

After Optimization:

```
a = a
```

🌟 Techniques Used

Technique	Purpose
Redundant Code Elimination	Removes useless instructions
Constant Folding	Computes constants at compile time
Strength Reduction	Replaces costly operations

🌟 Advantages

✅ Faster execution
✅ Reduced code size
✅ Better efficiency

🌟 Disadvantages

❌ Local optimization only
❌ Cannot optimize complete program

📌 Conclusion

Peephole Optimization improves instruction efficiency by modifying small instruction sequences.

🔥 Q6. Explain Register Allocation and Assignment.

✍️ Answer:

📌 Definition

Register Allocation assigns variables to CPU registers for faster execution.

Register Assignment assigns specific registers to variables.

🌟 Need of Register Allocation

Registers are:

```
Faster than memory
```

So compiler tries to use registers efficiently.

🌟 Example

```
R1 = a + b
```

🌟 Objectives

✅ Minimize memory access
✅ Improve execution speed
✅ Efficient register usage

🌟 Register Allocation Methods

Method	Description
Local Allocation	Inside one block
Global Allocation	Entire program

🌟 Advantages

✅ Faster execution

✅ Efficient memory usage

🌟 Disadvantages

❌ Limited number of registers

📌 Conclusion

Register Allocation improves program speed by efficient use of CPU registers.

🔥 Q7. Explain Boolean Expressions in Intermediate Code Generation.

✍️ Answer:

📌 Definition

Boolean Expressions produce:

```
True or False
```

values and are used in conditions and loops.

🌟 Example

```
if(a>b)
```

🌟 Relational Operators

Operator	Meaning
>	Greater than
<	Less than
==	Equal to

🌟 Intermediate Code Example

```
if a>b goto L1
goto L2
```

🌟 Applications

✅ If statements

✅ While loops

✅ Conditional expressions

🌟 Advantages

✅ Supports decision making

✅ Easy flow control

📌 Conclusion

Boolean Expressions are important for conditional branching during intermediate code generation.

🔥 Q8. Explain Issues in Design of Code Generator.

✍️ Answer:

📌 Definition

Code Generator converts intermediate code into machine code.

Its design must consider several important issues.

🌟 Major Design Issues

Issue	Description
Instruction Selection	Choosing proper instructions
Register Allocation	Efficient use of registers
Memory Management	Efficient storage use
Code Quality	Fast and optimized code
Target Machine	Hardware dependency

🌟 Objectives of Code Generator

✅ Generate efficient machine code

✅ Reduce execution time

✅ Minimize memory usage

🌟 Challenges

❌ Limited registers

❌ Machine dependency

❌ Optimization complexity

🌟 Advantages of Good Code Generator

✅ Faster execution

✅ Efficient memory use

✅ Better optimization

📌 Conclusion

Proper code generator design is essential for generating efficient and optimized machine code.

🔥 Q9. Explain Code Generation from DAG.

✍️ Answer:

📌 Definition

Code Generation from DAG uses Directed Acyclic Graph representation to generate optimized machine code.

🌟 Steps in Code Generation from DAG

1. Construct DAG

Represent expressions in graph form.

---

2. Identify Common Subexpressions

Remove repeated computations.

---

3. Generate Optimized Code

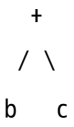
Produce efficient machine instructions.

🌟 Example

Statements:

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

DAG:



Single computation reused.

🌟 Advantages

✅ Eliminates redundancy

✅ Improves execution speed

✅ Reduces code size

🌟 Applications

✅ Compiler optimization

✅ Common subexpression elimination

📌 Conclusion

Code Generation from DAG produces efficient and optimized machine code by removing repeated computations.

⚡ EXAM WRITING TIPS FOR 14 MARKS

Always Include:

✅ Definition
✅ Diagram/Example
✅ Working
✅ Advantages
✅ Disadvantages
✅ Conclusion

🔥 MOST IMPORTANT QUESTIONS PRIORITY

1. Basic Block and Flow Graph
2. DAG Representation
3. Backpatching
4. Peephole Optimization
5. Intermediate Code Generation
6. Register Allocation
7. Code Generation from DAG