Devi Ahilya University, Indore, India Institute of Engineering & Technology				BE III Year Computer Engineering (FullTime)			
Subject Code : 6CERC2	Instructions Hours per Week			Credits			
Subject Name: Compiler Techniques	**L**	**T**	**P**	**L**	**T**	**P**	**TOTAL**
Duration of Theory Paper: 3 Hours	3	1	1	3	1	1	5

Learning Objectives:

- Understand the fundamentals of compiler design and the phases of compilation.
- Analyze compiler architecture and its major components.
- Develop skills in implementing and managing different compiler phases.
- Evaluate issues related to code optimization, correctness, and runtime efficiency.

Prerequisites: Computer Programming, Data Structures, Discrete Mathematics

Course Outcomes (CO) and Program Outcomes (PO) Mapping

CO No.	Course Outcome	Program Outcomes (PO)
CO1	Explain the fundamental concepts of compiler design	PO 1, PO 4, PO 11
CO2	Analyze the architecture and components of compilers	PO2, PO3, PO5, PO9
CO3	Apply techniques for designing and implementing compiler phases	PO6, PO10, PO10
CO4	Evaluate code optimization strategies and runtime efficiency	PO7, PO8, PO11,
CO5	Develop simple compiler modules or mini-compilers	PO2, PO4, PO5, PO10

CO-PO Relationship Matrix:

CO No.	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11
CO1	2	3	-	-	-	-	-	-	-	-	1
CO2	-	2	2	-	-	-	-	-	-	2	-
CO3	3	3	3	-	-	-	2	2	-	-	2
CO4	3	-	-	-	-	3	3	2	-	-	2
CO5	-	2	3	-	-	-	3	2	-	-	2

UNIT-I

Introduction to Compiling: Compilers: The Analysis-Synthesis Model of Compilation; Analysis of the Source Program: Lexical Analysis, Syntax Analysis, Semantic Analysis; The Phases of a Compiler: Symbol-Table Management, The Analysis Phases, Intermediate Code Generation, Code Optimization; Cousins of the Compiler: System Software, Interpreters, Preprocessors, Assemblers, Linkers and Loaders, Macros; The Grouping of Phases: Front and Back Ends, Passes; Compiler Construction Tools.

UNIT-II

Lexical Analysis: The Role of the Lexical Analyzer: Lexical Analysis Versus Parsing, Tokens, Patterns and Lexemes, Attributes for Tokens, Lexical Errors; Specification of Tokens: Strings and Languages, Regular Expressions; Recognition of Tokens: Transition Diagrams; Finite Automata: Nondeterministic Finite Automata, Deterministic Finite Automata; From Regular Expression to Automata: Conversion of an NFA to DFA, Construction of an NFA from a Regular Expression.

UNIT-III

Syntax Analysis: Introduction: The Role of the Parser, Classification of Grammars; Context-Free Grammars: Parse Tree and Derivations, Ambiguity, CFG Versus Regular Expressions; Writing a Grammar: Lexical Versus Syntactic Analysis, Eliminating Ambiguity; Top- Down Parsing: Recursive. Descent Parsing, LL(1) Grammars, Nonrecursive Predictive Parsing;Bottom-Up Parsing: Reductions, Shift Reduce Parsing; LR Parsing: Simple LR, Constructing SLR-Parsing Tables, Constructing LALR Parsing Tables.

UNIT-IV

Intermediate-Code Generation and Run-Time Environment: Variants of Syntax Trees:Directed Acyclic Graphs for Expressions; Three Address Code: Addresses and Instructions,Quadruples, Triples; Type Checking: Rules for Type Checking, Type Conversions; Storage Organization: Static Versus Dynamic Storage Allocation; Stack Allocation of Space: Activation Trees; Introduction to Garbage Collection.

UNIT-V

Code Optimization and Planning a Compiler: Basic Blocks and Flow Graphs: Basic Blocks, Flow Graphs; Optimization of Basic Blocks: The DAG Representation of Basic Blocks, Local Common Subexpressions Elimination, Semantic Preserving Transformations; Loop Optimization; Virtual Machines and their Interpreters; The LLVM Modular and Reusable Compiler Technologies, Overview of The HiPE Compiler; Case Studies of Compilers and Future Trends: The Sun Compilers for SPARC, The IBM XL Compilers for the POWER and POWERPC Architecture

Learning Outcomes:

Upon completing the course, students will acquire advance knowledge and understanding of Compiler. Use skills in Compiler Design. Apply acquired knowledge to improve performance of a Compiler. In addition to development in technology computer architectures offers a variety of resources of which students will be able to innovate in the Compiler Design.

Lab Components (Key Assignments)

1. Implement lexical analysis techniques to design and construct tokenizers using regular expressions, finite automata, and lexical analyzer generators.
2. Develop syntax analysis modules, including recursive descent parsers and table-driven parsers for LL and LR grammar constructs.

3. Generate intermediate code representations, such as three-address code, syntax trees, and DAG-based structures.
4. Perform type checking, symbol table management, and runtime storage allocation through practical implementation exercises.
5. Apply code optimization techniques on basic blocks and flow graphs, including dead-code elimination and loop optimization.
6. Integrate lexical, syntax, and semantic phases to build a mini-compiler or interpreter demonstrating end-to-end compilation.

Recommended Books:

[1] Compilers: Principles, Techniques, and Tools; Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman; Pearson Education.

[2] Compilers: Principles, Techniques, and Tools; Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman; Pearson Education.

[3] System Programming: D M Dhamdhere; McGraw Hill Education. [4] System Programming and Operating Systems: D M Dhamdhere; McGraw Hill Education.