Devi Ahilya University, Indore, India **Institute of Engineering & Technology**				**III Year B.E.** **Information Technology** **(Full Time)**			
Subject Code & Name	**Instructions Hours Per Week**			**Credits**			
	L	**T**	**P**	**L**	**T**	**P**	**Total**
ITR6G4 **Compiler Design**	**3**	**1**	**0**	**3**	**1**	**0**	**4**
Duration of Theory Paper:3 Hours							

Learning Objectives:

- Provide a framework for understanding the fundamentals of Compiler.
- To familiarize students with new Compilation Problems.
- Develop skills to understand how to Design a Compiler.
- Develop ability to understand how to enhance performance of a Compiler for newly evolved Programming Languages.

Prerequisite:

Knowledge of Computer Programming, Data Structures, Discrete Mathematics. Knowledge of several different Programming Languages will be useful.

COURSE CONTENTS

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 be:

- 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.

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.

Suggested Exercises:

For a Defined Language:

1. Design a symbol table mechanism.
2. Write an interpreter for quadruples.
3. Write the lexical analyser.
4. Write the semantic actions.

5. Write the parser.
6. Write the error-handling routines.
