Course Code: CSBB 351	**Open course (YES/NO)**	**HM Course (Y/N)**	**DC (Y/N)**	**DE (Y/N)**
	No	No	Yes	NO
Type of course	Core			
Course Title	**Compiler Design**			
Course Coordinator				
Course Objectives:	• To understand the structure and functions of a compiler. • To understand the various phases of compiler design. • To learn context-free grammar, compiler parsing techniques, and construction of abstract syntax trees. • To learn how to generate intermediate code by applying various code optimization strategies. • To gain hands-on experience designing and implementing a mini compiler through programming projects and case studies.			
Course Outcomes	**CO1:** Identify the phases of compilers for a programming language.			**L1, L2**
	CO2: Learn context-free grammars, compiler parsing techniques, construction of abstract syntax trees.			**L3**
	CO3: Generate syntax-directed translation rules for a given context-free grammar by examining S-attributed and L-attributed grammars and practice of compiler implementation and generate intermediate code by applying various code optimization strategies.			**L4**
	CO4: Build a domain-specific mini compiler.			**L5**

Semester	**Autumn: No**		**Spring: Yes**		
V	**Lecture**	**Tutorial**	**Practical**	**Credits**	**Total teaching hours**
Contact Hours	3	0	2	4	60
Prerequisite course code as per proposed course numbers					
Prerequisite credits					

Equivalent course codes as per proposed course and old course					
Overlap course codes as per proposed course numbers					
Text Books:					
1.	Title	Compilers Principles, Techniques, and Tools			
	Author	Alfred Aho, Ravi Sethi, Jeffrey D Ullman			
	Publisher	Pearson			
	Edition	2014			
Reference Book:					
1.	Title	The Theory and Practice of Compiler Writing			
	Author	Tremblay, Sorenson			
	Publisher	Addison-Wesley			
	Edition	1992			
2.	Title	Compiler Design in C			
	Author	Holub			
	Publisher	PHI			
	Edition				

Content	**Unit 1** **Introduction**: Compilers Analysis of the source program, Phases of a compiler, Cousins of the Compiler, Grouping of Phases, Bootstrapping and Compiler construction tools, Symbol Table. **Lexical Analysis:** Role of Lexical Analyzer, Input Buffering, Specification of Tokens, Recognition of Tokens, From Regular expression to Automata and Design of Lexical Analysis generator. **Unit 2** **Syntax Analysis**: Role of the parse, Writing Grammars, Context-Free Grammars, Ambiguous Grammars, Top-Down parsing, Recursive Descent Parsing, Predictive Parsing, Bottom-up parsing, Shift Reduce Parsing, Operator Precedence Parsing, LR Parsers, SLR Parser, Canonical LR Parser, LALR Parser. **Unit 3** **Syntax Directed Translation:** Syntax Directed Definitions, Application of SDT and SDT schemes. **Unit 4** **Intermediate Code Generation:** Directed acyclic graphs, three-address code Intermediate languages - Declarations, Assignment Statements, Boolean Expressions, Array references, Back patching. **Unit 5** **Code Generation and Optimization:** Issues, Basic Blocks, and Flow Graphs, DAG representation of Basic Blocks, Optimization of basic Blocks, Peephole Optimization, Principal Sources of Optimization, Loop Optimization, Global Data Flow Analysis.

List of Experiments	1. Identify the files created during the compilation and execution of a C program. 2. Create a program to generate a Lexical Analyzer to identify the following tokens: Keywords, Identifier, Operators, Digits, Literals, and Separators. 3. Read about the FLEX tool and its installation, and then create a program using FLEX to generate a Lexical Analyzer and identify the following tokens: Keywords, Identifier, Operators, Literals, and Separators. 4. Write a program using FLEX to count the number of Lines, Words, Capital Letters, Small Letters, Numbers, Digits, Special Character, Delimiter, Operator, Relational Operator, Total Characters. 5. Write a program using FLEX to read and validate a mathematical expression and display the result. The result should follow the BODMAS Rule. If the expression is invalid, then display it as invalid. 6. Mini Project: Domain/Application Specific Project: Create a lexical analyzer for the Mini Project 7. Perform the following: a. Write a C program to find First and Follow for all non-terminals. b. Explore YACC Tool. c. Download and Install the YACC Tool on your system and prepare an installation lab manual. 8. Perform the following: a. Write a YACC program to check if the entered statement is a valid arithmetic expression. b. Design a grammar for a declarative statement for the C program. Further, write a YACC program to check if the entered statement is a valid declarative statement according to the grammar generated. c. Design a grammar for a relational expression of C language. Further, write a YACC program to check if the entered statement is a valid relational expression according to the grammar generated. 9. Perform the following: a. Consider the example of a simple desk calculator. You are required to write YACC specifications for this grammar. Show your parsing sequence for the sample input string. b. Write a YACC program to recognize a series of expressions, each on a new line. Also, give a parse sequence for each of the expressions. c. For the grammar below, write a YACC program to compute the decimal value of an input string in binary. 10. Mini Project: Domain/Application Specific Project Create syntax analyzer for the Mini Project 11. Perform the following: a. Using YACC, write semantic actions to check if the parenthesis is balanced and count the number of matching parenthesis. P->(P) | a b. Using YACC, write semantic actions translating arithmetic expressions from infix into postfix notation. 12. Perform the following: a. Append semantic actions to the grammar of a simple desk calculator. b. Write a YACC program to evaluate sum, product, min, and max expressions. 13. Write a program to generate machine code (Assembly Code) from an abstract syntax tree generated by the parser.

Course Assessment	Continuous Evaluation 25%
	Mid Semester 25%
	End Semester 50%