COMPILER DESIGN

V Semester: CSE / IT									
Course Code	Category	Hours / WEEK			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSB11	Core	2	1	-	3	30	70	100
Contact Classes: 30	Tutorial Classes: 15	Practical Classes: Nil				Total Classes:45		

I. COURSE OVERVIEW:

This course describes the basic techniques for compiler construction and tools that can be used to perform syntax-directed translation of a high-level programming language into an executable code. It will provide deeper insights into the more advanced semantics aspects of programming languages, machine independent optimizations and code generation.

II. OBJECTIVES:

The course should enable the students to:

I	The process of translating a high-level language to machine code required for compiler construction.
II	The Software tools and techniques used in compiler construction such as lexical analyzer and parser generators.
III	The data structures used in compiler construction such as abstract syntax trees, symbol tables, three-address code, and stack machines.
IV	The deeper insights into the syntax and semantic aspects of programming languages, dynamic memory allocation and code generation.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO 1	**Summarize** phases of a compiler in the construction of language processors.	Understand
CO 2	**Make use of** finite automata for designing a lexical analyzer for a specific programming language constructs.	Apply
CO 3	**Choose** top down, bottom up parsing methods for developing a parser with representation of a parse table or tree.	Apply
CO 4	**Outline** syntax directed translations, intermediate forms for performing semantic analysis along with code generation.	Understand
CO 5	**Relate** symbol table, type checking and storage allocation strategies used in run-time environment.	Understand
CO 6	**Select** code optimization techniques on intermediate code form for generating target code.	Apply

IV. SYLLABUS:

MODULE-I	INTRODUCTION TO COMPILERS	Classes: 08

Introduction to compilers**:** Definition of compiler, interpreter and its differences, the phases of a compiler; Lexical Analysis: Role of lexical analyzer, input buffering, recognition of tokens, finite automata, regular Expressions, from regular expressions to finite automata, pass and phases of translation, bootstrapping, LEX-lexical analyzer generator.

MODULE-II	SYNTAX ANALYSIS	Classes: 09

Syntax Analysis: Parsing, role of parser, context free grammar, derivations, parse trees, ambiguity, elimination of left recursion, left factoring, eliminating ambiguity from dangling-else grammar; Types of parsing: Top-down parsing, backtracking, recursive-descent parsing, predictive parsers, LL (1) grammars. Bottom-up parsing: Definition of bottom-up parsing, handles, handle pruning, stack implementation of shift-reduce parsing, conflicts during shift-reduce parsing, LR grammars, LR parsers-simple LR, canonical LR and Look Ahead LR parsers, YACC-automatic parser generator.

MODULE-III	SYNTAX-DIRECTED TRANSLATION AND INTERMEDIATE CODE GENERATION	Classes: 10
Syntax-Directed Translation: Syntax directed definitions, construction of syntax trees, S-attributed and L-attributed definitions; Syntax Directed Translation schemes. Intermediate code generation: Intermediate forms of source programs– abstract syntax tree, polish notation and three address code, types of three address statements and its implementation, syntax directed translation into three-address code, translation of simple statements, Boolean expressions and flow-of-Control statements.		
MODULE-IV	**TYPE CHECKING ANDRUN TIME ENVIRONMENT**	**Classes: 09**
Type checking: Definition of type checking, type expressions, type systems, static and dynamic checking of types, specification of a simple type checker; Run time environments: Source language issues, Storage organization, storage-allocation strategies, access to nonlocal data on the stack, garbage collection, symbol tables.		
MODULE-V	**CODE OPTIMIZATION AND CODE GENERATION**	**Classes: 09**
Code optimization: The principle sources of optimization, optimization of basic blocks, loops in flow graphs, peephole optimization; Code Generation: Issues in the Design of a Code Generator, The Target Language, addressesin the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, A Simple Code Generator, register allocation and assignment, DAG representation of basic blocks.		

Text Book:

1. Alfred V.Aho, Ravi Sethi, Jeffrey D, Ullman, “Compilers–Principles ,Techniques and Tools”, Pearson Education, 2nd Edition, 2006.

Reference Books:

1. Kenneth C.Louden,Thomson, “Compiler Construction–Principles and Practice”, PWS Publishing, 1st Edition, 1997.
2. Andrew W. Appel, “Modern Compiler Implementation C”, Cambridge University Press, Revised Edition, 2004.

Web References:

1. www.vssut.ac.in/lecture_notes/lecture1422914957.pdf
2. http://csenote.weebly.com/principles-of-compiler-design.html
3. http://www.faadooengineers.com/threads/32857-Compiler-Design-Notes-full-book-pdf-download
4. https://www.vidyarthiplus.com/vp/thread-37033.html#.WF0PhlMrLDc

E-Text Books:

1. http://www.e-booksdirectory.com/details.php?ebook=10166
2. http://www.e-booksdirectory.com/details.php?ebook=7400re