SOURCE CODE

The xv6 source code is available via : *git clone git://pdos.csail.mit.edu/xv6/xv6.git*

EVALUATION METHOD TO BE USED:

Category of Course	Continuous Assessment	Mid – Semester Assessment	End Semester
Theory Integrated with Practical	**15(T) + 25 (P)**	**20**	**40**

CO - PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	✓	✓					✓				✓	✓
CO2	✓	✓		✓	✓	✓					✓	✓
CO3	✓	✓	✓	✓	✓		✓				✓	✓
CO4	✓	✓	✓	✓	✓	✓					✓	✓
CO5	✓	✓				✓	✓				✓	✓

CS 6109 COMPILER DESIGN

OBJECTIVES:

- To know about the various transformations in the different phases of the compiler, error handling and means of implementing the phases
- To learn about the techniques for tokenization and parsing
- To understand the ways of converting a source language to intermediate representation
- To have an idea about the different ways of generating assembly code
- To have a brief understanding about the various code optimization techniques

COMPILER DESIGN	L	T	P	EL	CREDITS
	3	0	4	3	6

MODULE I :	L	T	P	EL
	3	0	4	3
Phases of the compiler – compiler construction tools – role of assemblers, macroprocessors, loaders, linkers.				
SUGGESTED ACTIVITIES : • EL – Constructs of programming languages - C, C++, Java • LEX tool tutorial				
SUGGESTED EVALUATION METHODS: • Tutorial problems				

- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE II :	L	T	P	EL
	3	0	4	3

Role of a lexical analyzer – Recognition of Tokens – Specification of Tokens - Finite Automata (FA) – Deterministic Finite Automata (DFA) – Non-deterministic Finite Automata (NFA) – Finite Automata with Epsilon Transitions – NFA to DFA conversion - Minimization of Automata.

SUGGESTED ACTIVITIES :

- EL –LEX tool for tokenization
- Problems based on conversion from NFA to DFA, Epsilon NFA to DFA
- Practical – Programs using LEX for tokenization

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE III :	L	T	P	EL
	3	0	4	3

Error handling – Error Detection and Recovery – Lexical phase error management – Syntax phase error management -Error recovery routines.

SUGGESTED ACTIVITIES :

- Flipped Class room – LEX programs
- Problems based on obtaining automata for error routines.
- EL – Implementation of error recovery procedures using LEX/FLEX tool

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE IV :	L	T	P	EL
	3	0	4	3

Context-Free Grammar (CFG) – Derivation Trees – Ambiguity in Grammars and Languages – Need and Role of the parser

SUGGESTED ACTIVITIES :

- EL - CFG for C language constructs
- Problems to check for ambiguity

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes

MODULE V :	L	T	P	EL
	3	0	4	3

Recursive Descent Parsers – LL(1) Parsers – Shift Reduce Parser – LR(0) items - Simple LR parser

SUGGESTED ACTIVITIES :

- EL – Push down automata for Parsing, YACC tutorial.
- Problems based on simplification of CFG

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes

MODULE VI:	L	T	P	EL
	3	0	4	3

LALR Parser – CALR Parser – Parser Generators – Design of a parser generator

SUGGESTED ACTIVITIES :

- EL – YACC tutorial for parsing particular language syntaxes
- Practical – programs using YACC for parsing

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE VII:	L	T	P	EL
	3	0	4	3

Syntax directed Definitions – Inherited and Synthesized Attributes - Syntax Directed Translation - Construction of Syntax Tree-Type Systems-Specification of a simple type checker

SUGGESTED ACTIVITIES :

- EL – Type checking semantic rules for a programming language like C.
- Programs for validating C-lite constructs using YACC

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes

MODULE VIII:	L	T	P	EL
	3	0	4	3

Three address code – Types of Three address code – Quadruples, Triples, Three-address code for Declarations, Arrays, Loops, Backpatching

SUGGESTED ACTIVITIES :

- Flipped classroom – semantic rules for three-address code a programming language like C.
- Practical – implementation of three-address code generation for a programming language like C.
- EL – Three-address code for Switch-case statements
- Assignment on generating three-address code for arrays, looping constructs with and without backpatching

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE IX:	**L**	**T**	**P**	**EL**
	3	**0**	**4**	**3**

Run Time Environment: Source Language Issues- Symbol Tables - Storage Organization-Stack Allocation- Access to nonlocal data on stack – Heap management - Parameter Passing

SUGGESTED ACTIVITIES :

- Flipped classroom – suggested parameter passing techniques for a programming language like C.
- Practical – Symbol table implementation

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Practical demo / evaluation

MODULE X:	**L**	**T**	**P**	**EL**
	3	**0**	**4**	**3**

Basic blocks – Next use – Register allocation – DAG construction – Loops

SUGGESTED ACTIVITIES :

- Combination of in class & Flipped
- EL – Basic block, next-use applications,
- EL – alternate register allocation techniques
- Practical – Implementation of Register allocation using Graph colouring

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE XI:	**L**	**T**	**P**	**EL**
	3	**0**	**4**	**3**

Code Generator Issues – Simple Code generator – Data Structures for simple code generator, Labelling algorithm - Code generator using DAG – Dynamic programming based code generation

SUGGESTED ACTIVITIES :

- Combination of in class & Flipped
- EL – Template based code generation
- Practical – simple code generator for a programming language like C.

SUGGESTED EVALUATION METHODS:

- Tutorial problems
- Assignment problems
- Quizzes
- Practical demo / evaluation

MODULE XII:	L	T	P	EL
	3	0	4	3
Principle sources of optimization - Optimization in Basic blocks – DAG – Structure Preserving transformation – functional transformation – loop optimization – Peep hole optimization				
SUGGESTED ACTIVITIES : • Combination of in class & Flipped • Practical – Combining and integrating all the implemented features for a programming language like C				
SUGGESTED EVALUATION METHODS: • Tutorial problems • Assignment problems • Quizzes • Practical demo / evaluation				

TEXT BOOK:

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, “Compilers: Principles, Techniques and Tools”, Second Edition, Pearson Education Limited, 2014.

REFERENCES:

1. Randy Allen, Ken Kennedy, “Optimizing Compilers for Modern Architectures: A Dependence-based Approach”, Morgan Kaufmann Publishers, 2002.
2. Steven S. Muchnick, “Advanced Compiler Design and Implementation”, Morgan Kaufmann Publishers - Elsevier Science, India, Indian Reprint, 2003.
3. Keith D Cooper and Linda Torczon, “Engineering a Compiler”, Morgan Kaufmann Publishers, Elsevier Science, 2004.
4. V. Raghavan, “Principles of Compiler Design”, Tata McGraw Hill Education Publishers, 2010.
5. Allen I. Holub, “Compiler Design in C”, Prentice-Hall Software Series, 1993.

OUTCOMES:

Upon completion of the course, the students will be able to:

- Comprehensively identify the issues in every phase of the compiler
- Analyse the design issues in the different phases of the compiler and design the phases by integrating appropriate tools
- Identify the apt code generation strategy that needs to be adopted for any given source language
- Analyse and understand the various code optimizations that are necessary for any given intermediate code or assembly level code for sequential algorithms
- Apply and design code optimization techniques for any input code with error recovery
- Design a compiler by incorporating the various phases of the compiler for any new source language

Category of Course	Continuous Assessment	Mid – Semester Assessment	End Semester
Theory Integrated with Practical	15(T) + 25 (P)	20	40

CO - PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	✓	✓	✓									✓
CO2	✓	✓	✓	✓	✓	✓					✓	✓
CO3	✓	✓	✓	✓	✓	✓	✓				✓	✓
CO4	✓	✓	✓	✓	✓	✓					✓	✓
CO5	✓	✓	✓	✓	✓	✓	✓				✓	✓
CO6	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓

CS6110 OBJECT ORIENTED ANALYSIS AND DESIGN

Prerequisites for the course: None

OBJECTIVES:

- To capture the requirements specifications of an intended software system
- To design software with static and dynamic UML diagrams
- To map the design properly to code
- To improve the software design with design patterns
- To test the software against its requirements specifications

OBJECT ORIENTED ANALYSIS AND DESIGN	L	T	P	EL	CREDITS
	3	0	4	3	6

MODULE I :	L	T	P	EL
	3	0	4	3

Introduction to OOAD with OO Basics - Unified Process – UML diagrams

SUGGESTED ACTIVITIES :

- EL - Identifying a suitable case study to work on for a complete end-end implementation
- EL – Document the Software Requirement Specifications(SRS) for the identified case study
- Practical – Getting familiar with the case tool

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Quizzes