Lecture 1 - Introduction and Course Outline
Compiler Design (CS 3007)

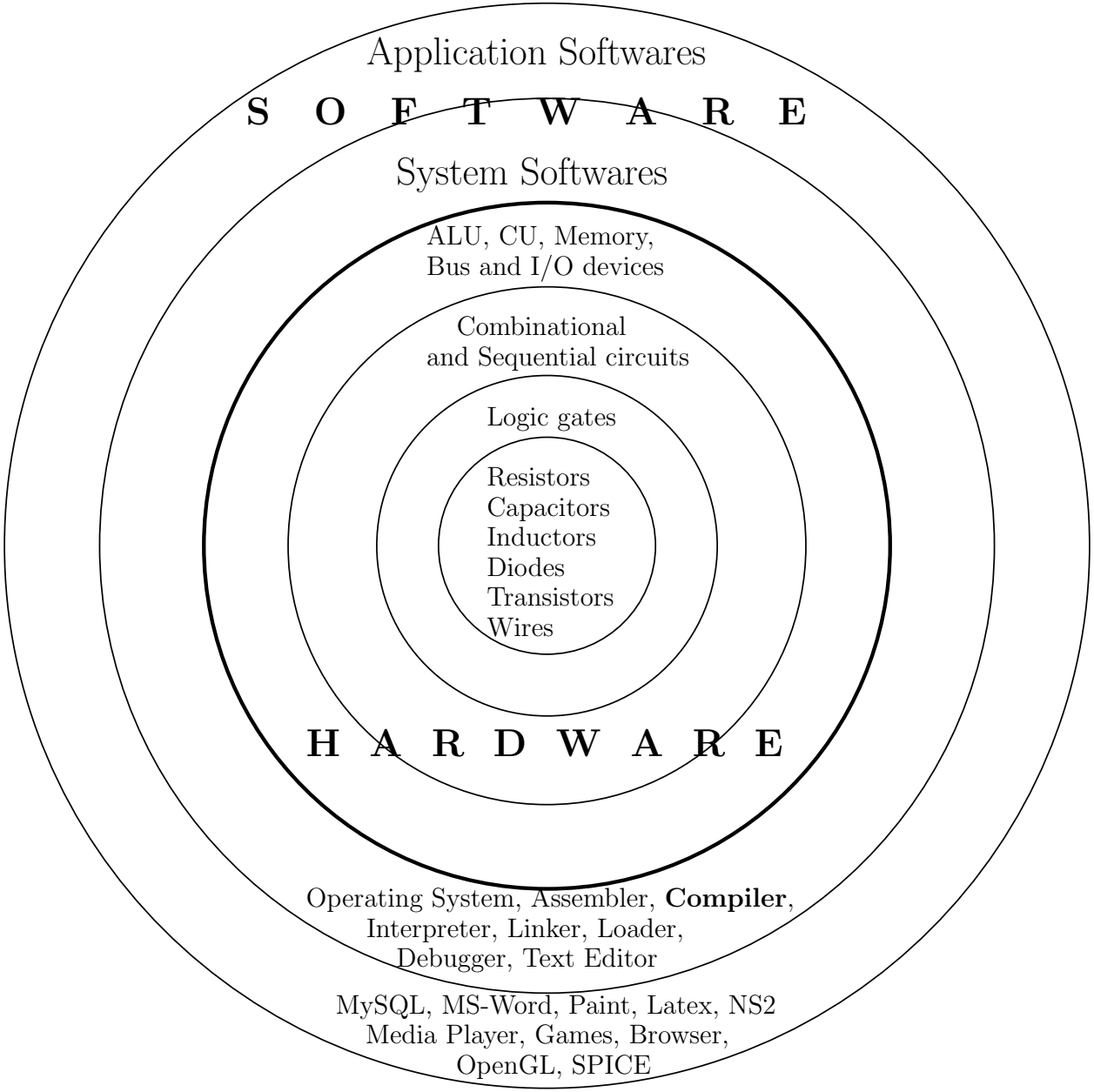
S. Pyne[1] & T. K. Mishra[2]

Assistant Professor[1,2]
Computer Science and Engineering Department
National Institute of Technology Rourkela
{pynes,mishrat}@nitrkl.ac.in

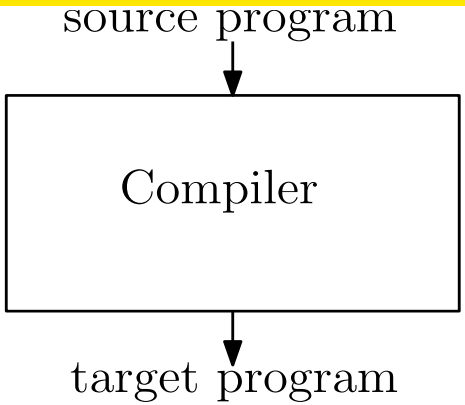
August 4, 2020

Overview

Hardware and Software in Modern Computers

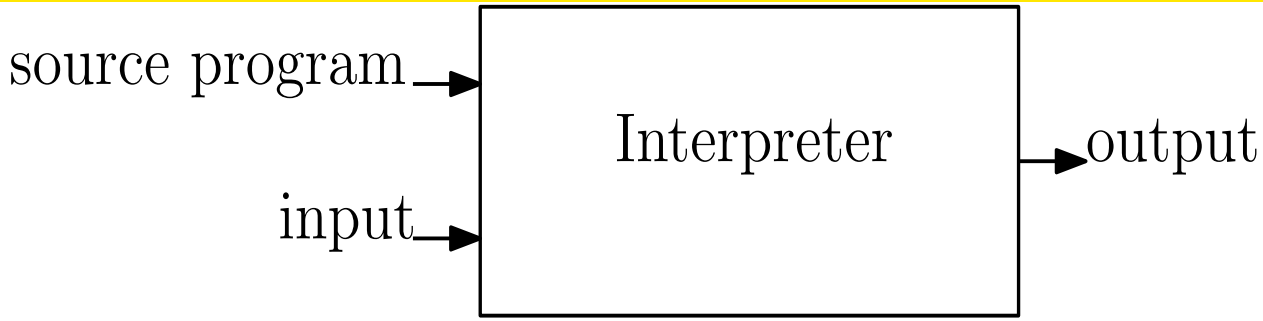


What is a Compiler ?



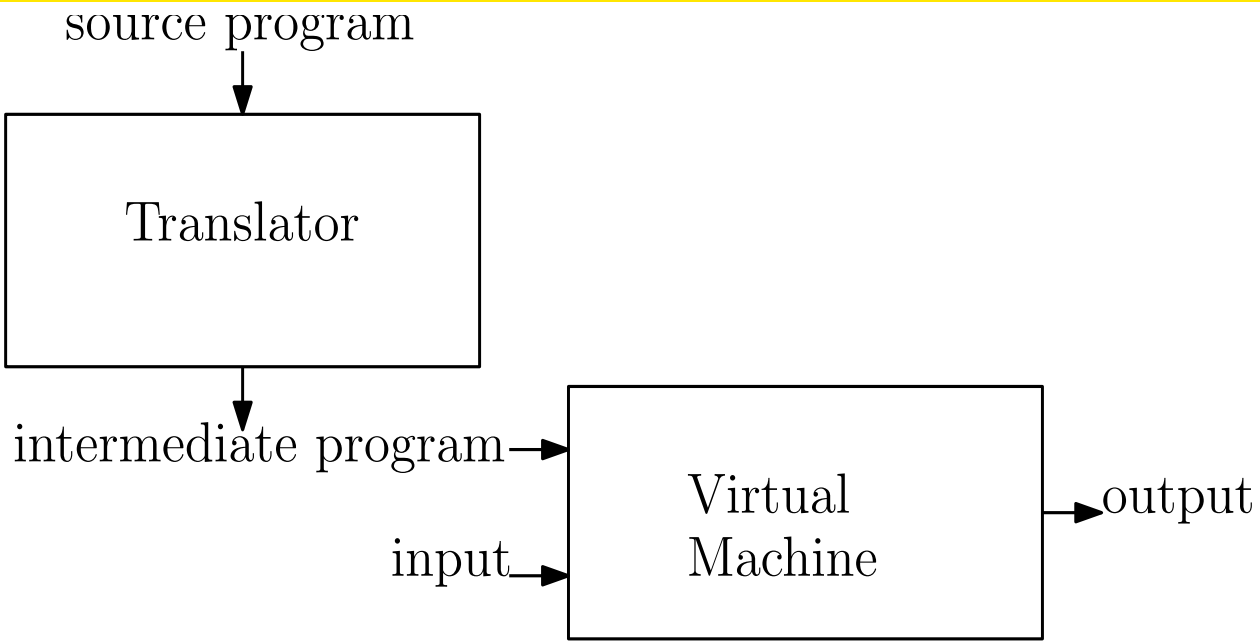
- A system software
- Converts a source program to a target program
- Source program - high level language like C, C++, Pascal, etc.
- Target program
  - usually an assembly language or a machine level language (native code) for an instruction set architecture (ISA)
  - may be another high level language in case of laguage converters like C to Fortran
- Examples
  - The GNU Compiler Collections (GCC)
  - Java programming language compiler (JAVAC)

Interpreter - An alternative to compiler



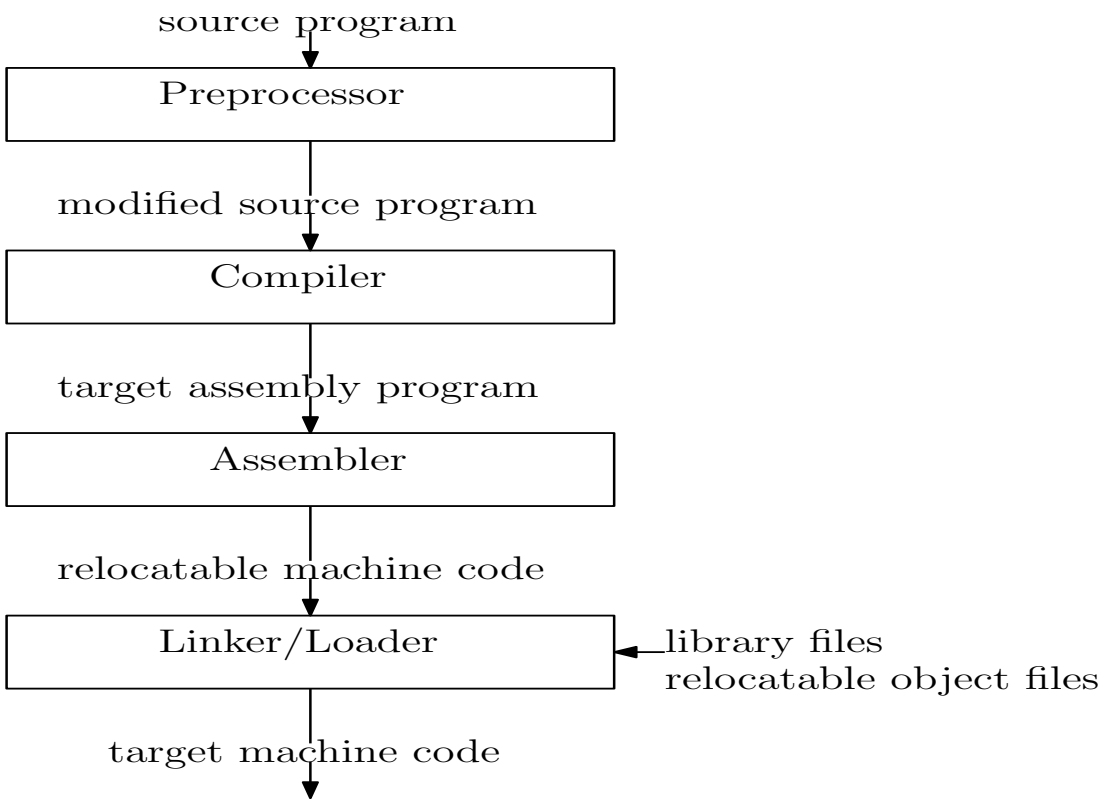
- Runs a program by converting (interpreting) each high-level statement of the program into machine code
- Unlike compilers, no need to generate and save of target code
- Program is interrupted for each erroneous statement encountered which needs to be corrected to interpret the next statement
- For compilers, an error free program can run only after successful generation of target program
- Programming languages like PHP, Perl, Ruby uses interpreter
- Interpreted programs run slower than compiled programs
- Interpreter provides platform indepedence
- Example - JVM interpretes byte code generated by JAVAC

Compiler and Interpreter



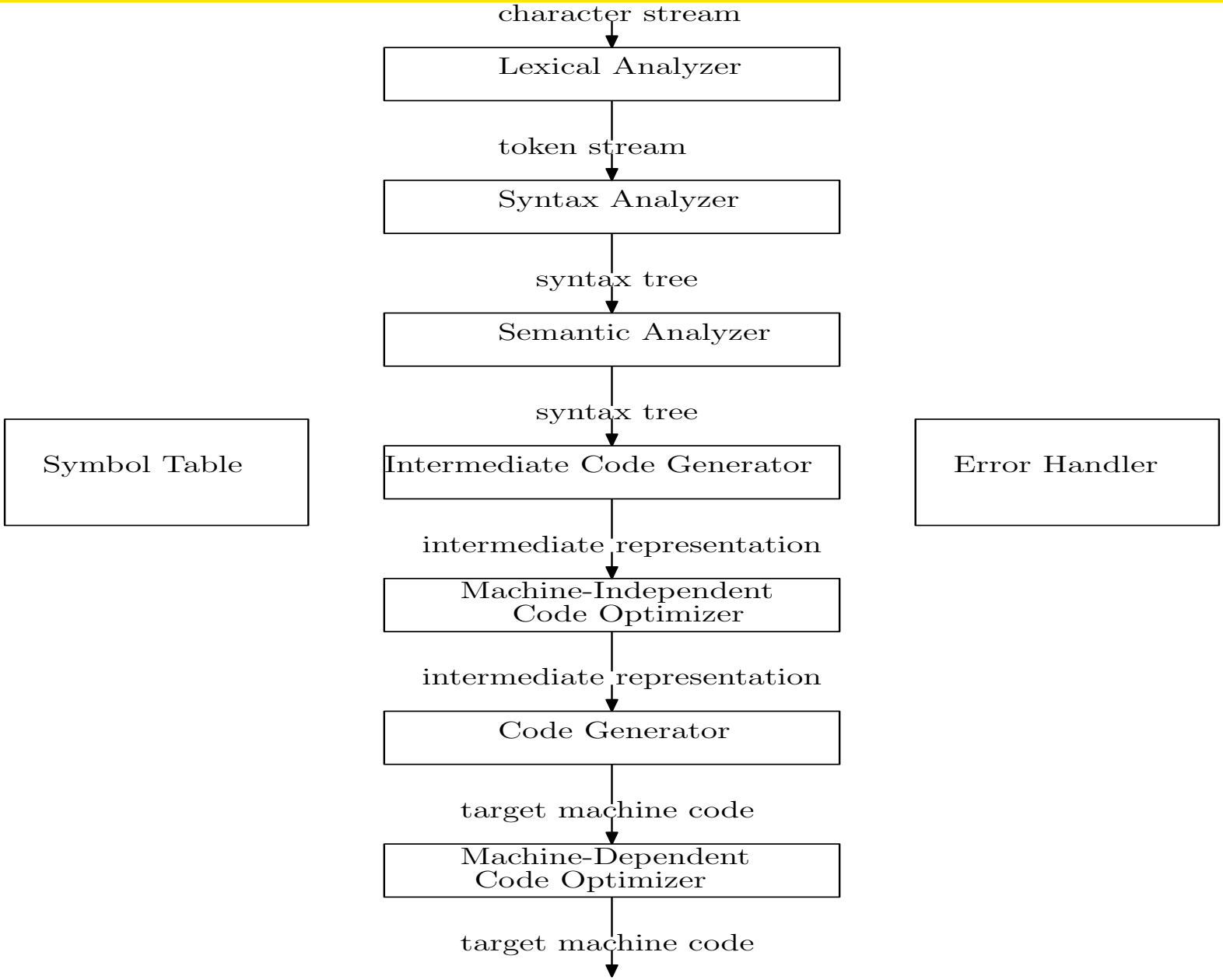
- Compiler generates intermediate code which is run by interpreter
- Intermediate code is platform independent
- For example
  - JAVAC compiles a java program to generate byte code in class files
  - JVM interpretes the byte codes to run the program
  - Java program independent of target hardware and operating system
  - Compile once and run in any machine with JVM

Language processing system



- Preprocessing substitute macros with symbolic constants
- Compiler generates target assembly program
- Assembler produces machine code
- Linker links the program with library files
- Loader - part of operating system that initiates program execution

Phases of Compilation



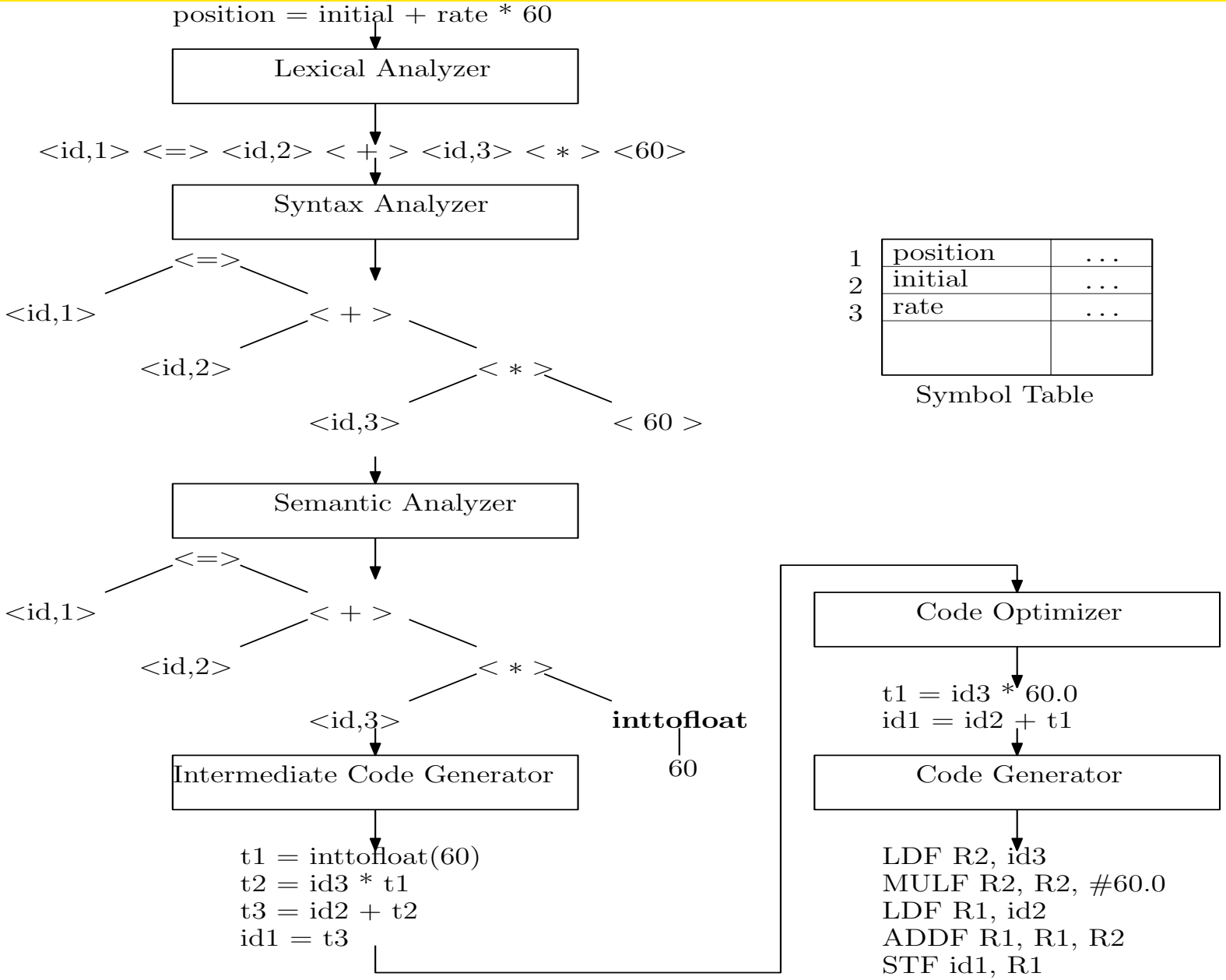
Phases of Compilation

- Preprocessing - Text replacement for substitution of macro definitions.
- Lexical Analysis - Recognize tokens like keywords, identifiers, constants, etc.
- Syntax Analysis - Parsing to check grammatical correctness of the statements.
- Semantic Analysis - Check meaning of the statements. Does type checking and type conversion.
- Intermediate code generation - Generates a machine independent code which can be easy converted to assembly code of target machine
- Target code generation - Converts intermediate code to assembly code of target machine with register allocation
- Code Optimization - Machine independent optimization for intermediate code (loop optimization) and machine dependent optimization for target code (replace "MUL A,2" with "LSHF A")

Other activities of a Compiler

- Symbol Table Management - Track names used by the program and records essential information about each such as type (int, float, etc.).
- Error Handling - Report undefined variables, syntax error, operations with incompatible types, unreachable statements, constant exceeding word length of target machine, multiple declaration of an identifier.
- Run-time Storage Administration - Stack and heap management for local, global and static variables, procedure call and return, activation record for procedures, parameter passing in procedures, creation of array space, scope of identifiers, etc.

Translation

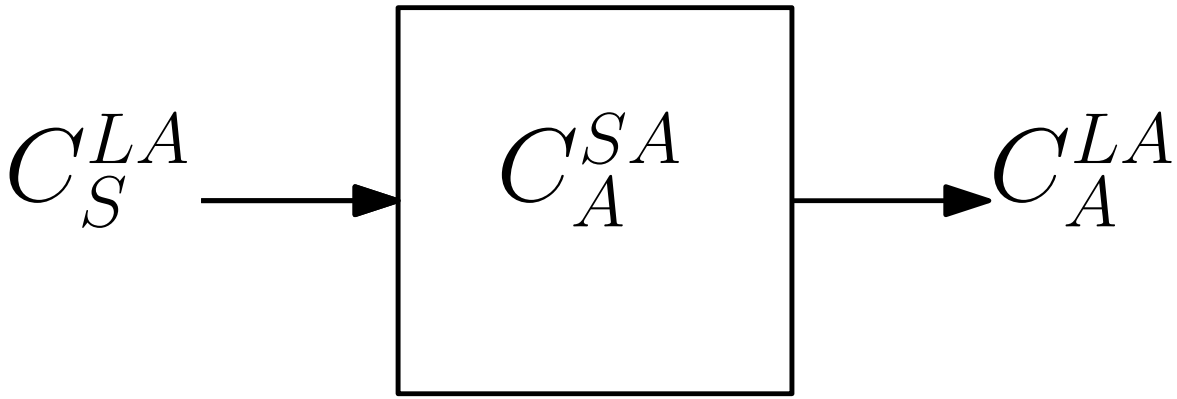


Passes in Compiler

- Number of scans done on a program and its equivalent
- Single pass - All phases in one pass
- Multipass - Slower and requires less memory
- A two pass compiler
  - Pass 1 - lexical analysis, syntax analysis, semantic analysis and intermediate code generation
  - Pass 2 - code optimization and target code generation
- For a program with following statements
  GOTO L
  ⋮
  L: ADD X
  - A two-pass assembler in 1^{st} pass makes a symbol table entry for L, replaces GOTO and L with opcode and jump address in 2^{nd} pass
  - A one-pass assembler on encountering "L: ADD X" scans the list of statements referring to L and places the address of "L: ADD X" to the address field of "GOTO L"
- Backpatching - Merging of phases into one pass

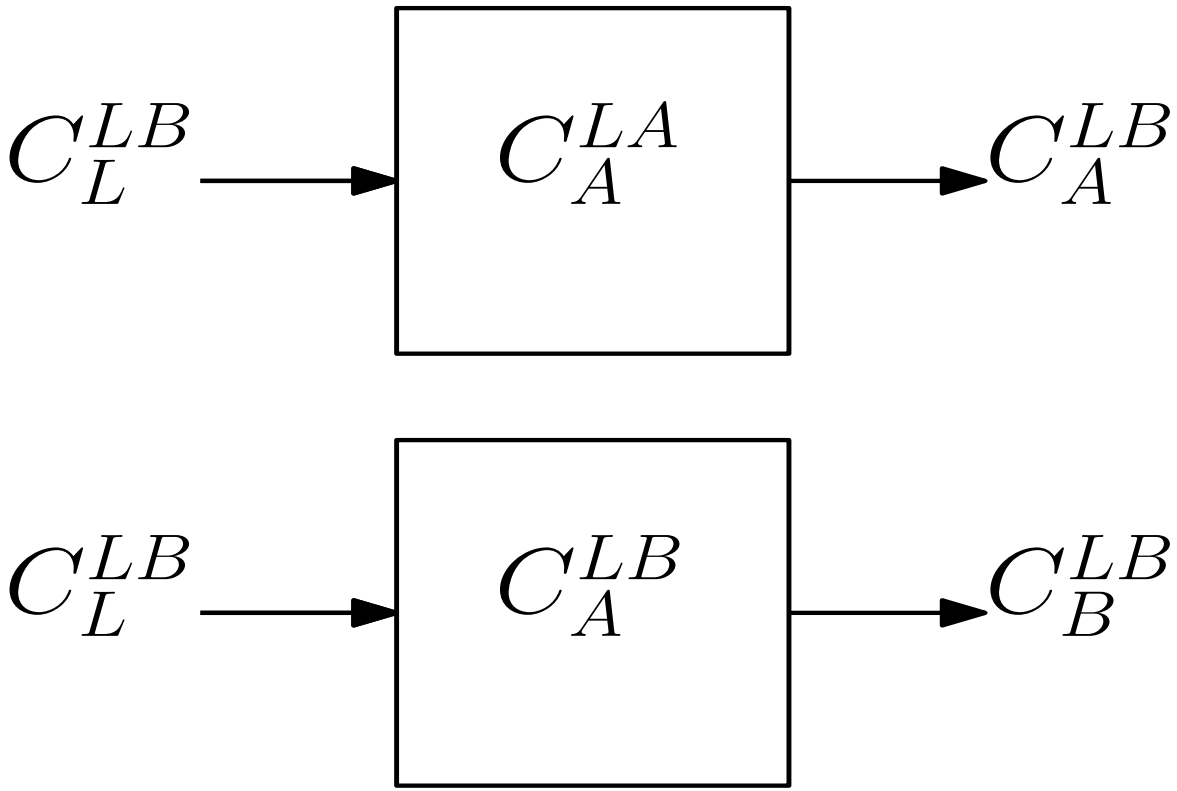
Bootstrapping

- How was the first compiler compiled ?
- A compiler C_Z^{XY} requires three languages
  - Source language (X) - Language of source program
  - Object language (Y) - Language of target program
  - Implementation language (Z) - Language used to write the compiler
- Bootstrapping C_A^{LA} for a new language L with target machine A
  - First write C_A^{SA} that translates $S \subset L$ to language of A
  - Then write C_S^{LA} and compile it through C_A^{SA} to get C_A^{LA}

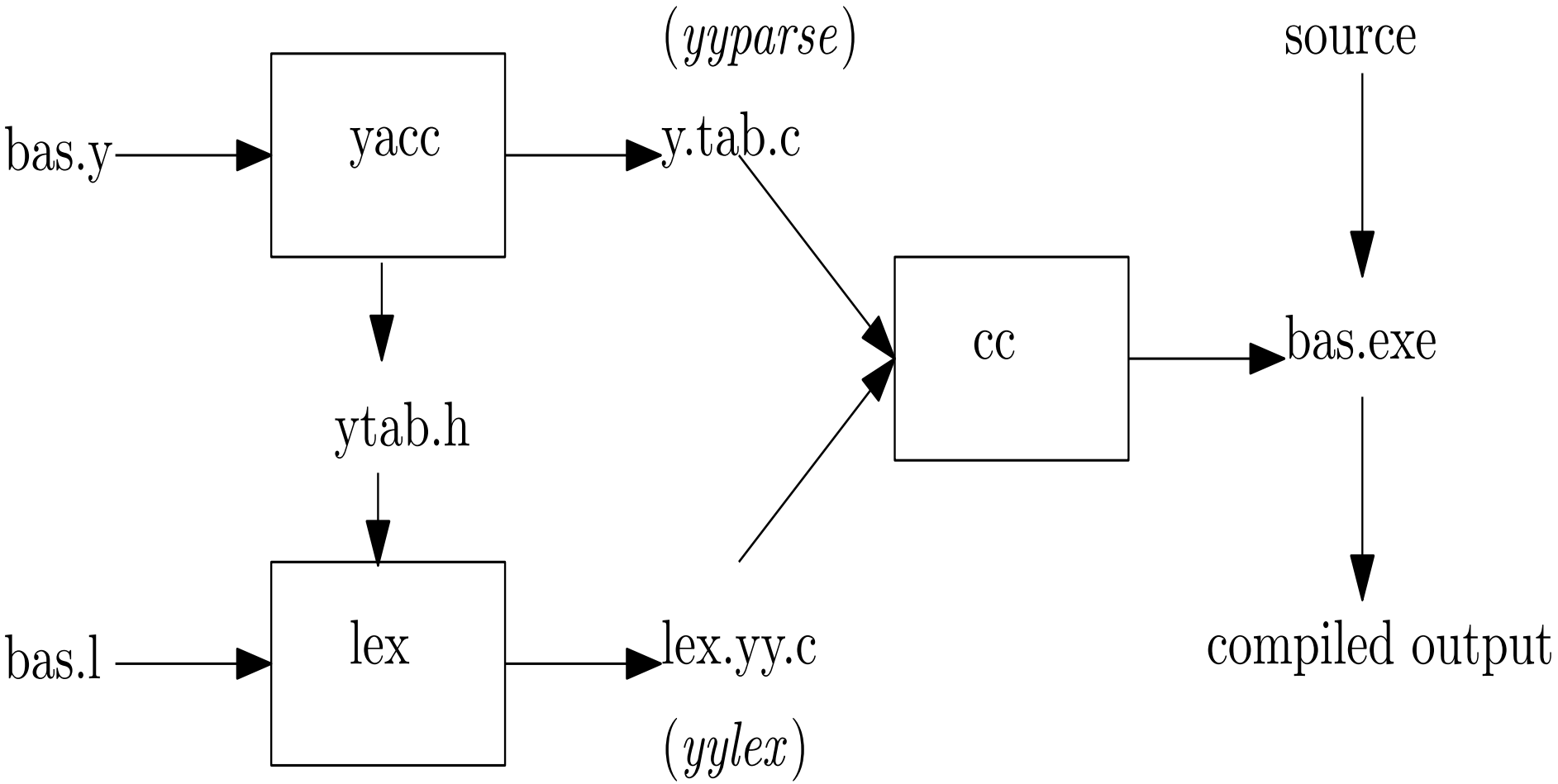


Bootstrapping and Cross Compiler

- Design compiler C_B^{LB} for L to run on machine B using machine A
- First write C_L^{LB} and compile it with C_A^{LA} to get C_A^{LB}
- Then compile C_L^{LB} with C_B^{LA} to get C_B^{LB}
- C_A^{LB} is a cross compiler
- C_A^{LB} runs on machine A to produce code for machine B



Building a Compiler with Lex and Yacc



Syllabus

1. Lexical analysis - Tokens, regular expressions, NFA, DFA, automatic construction of lexical analyzer, data structures of lexical analyzer, symbol table entry, Lex tool.
2. Syntax analysis - CFG, PDA, top-down parsing (brute force, recursive descent, LL(1)), bottom-up parsing (operator-precedence, shift-reduce, LR(0), SLR(1), CLR(1), LALR(1)), Yacc tool.
3. Syntax directed translation / Semantic Analysis - Semantic rules and actions, parse tree evaluation, translation to postfix and 3-address codes.
4. Intermediate Code Generation - Postfix notations, syntax trees, quadruples and triples for assignment, control statements and array.
5. Register allocation and code generation - Register allocation using labelled tree and graph coloring methods.
6. Code optimization - basic blocks, flow graphs, loop optimization, code motion, strength reduction, elimination of loop invariant variables and unreachable code, machine dependent optimization.
7. Runtime storage management- memory allocation variables and arrays, stack and heap memory management, parameter passing, activation records.
8. Symbol table management - data structures for symbol table; insert, delete, search and update operations.
9. Error handling - undefined identifiers, multiple declaration, syntax and semantics errors, incompatible types.

Text and Reference Books

- Essential Reading
  - Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, Compilers - Principles, Techniques, and Tool, 2^{nd} Edition, Pearson.
  - John Levine, Unix Text Processing Tools - Flex and Bison, O'REILLY.
- Supplementary Reading
  - Allen I. Holub, Compiler Design in C, Prentice Hall.

Examinations and Marks Distribution

- Class Test 1 – 10%
- Mid Semester – 30%
- Class Test 2 – 10%
- End Semester – 50%

Prerequisites

- Programming in C/C++/Java/Python
- Data Structures - arrays, linked lists, trees, graphs, hashing
- Theory of Computation - Finite automata and push down automata
- Computer architecture and assembly language

Thank you