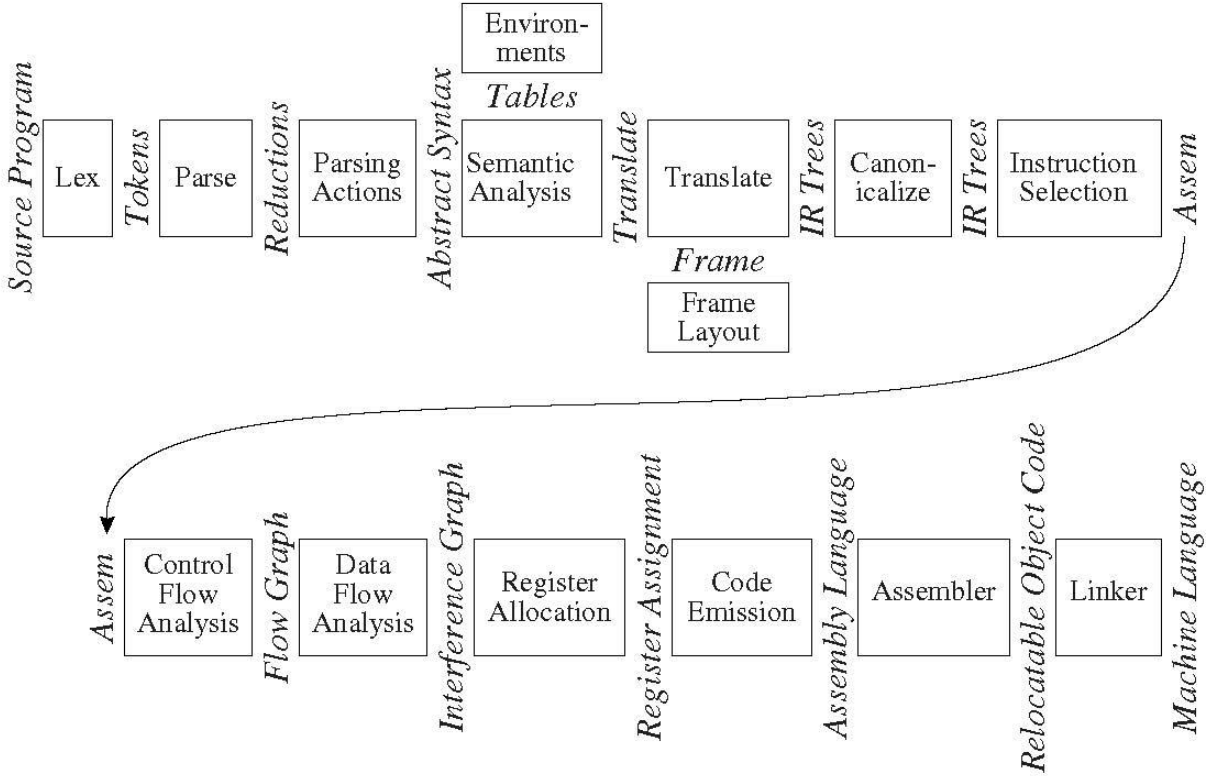
Compiler phases

Prof Rajeev Barua

Program Analysis -- Slide Set 3

Phases of a modern compiler



IGURE 1.1. Phases of a compiler, and interfaces between them.

Phases for other Program Analysis tools are some subset of the above passes.

- Instruction selection is usually after Data Flow analysis.
- The phases from Lex to Canonicalize are called the **Compiler Front End**. (Changed for each language).
- The phases from Instruction selection, register allocation, and onwards until code emission are called the **Compiler back end**. (changed for each new ISA).
- The phases in the middle are the **Compiler optimizations**. These improve the IR code.
- The Assembler is only needed if assembly code (which is human readable) is needed to be produced.

Explanation of compiler phases

PLAKSHA UNIVERSITY

Chapter	Phase	Description
2	Lex	Break the source file into individual words, or *tokens*.
3	Parse	Analyze the phrase structure of the program.
4	Semantic Actions	Build a piece of *abstract syntax tree* corresponding to each phrase.
5	Semantic Analysis	Determine what each phrase means, relate uses of variables to their definitions, check types of expressions, request translation of each phrase.
6	Frame Layout	Place variables, function-parameters, etc. into activation records (stack frames) in a machine-dependent way.
7	Translate	Produce *intermediate representation trees* (IR trees), a notation that is not tied to any particular source language or target-machine architecture.
8	Canonicalize	Hoist side effects out of expressions, and clean up conditional branches, for the convenience of the next phases.

Figure 1-2. Top part

Explanation of compiler phases (continued)

9	Instruction Selection	Group the IR-tree nodes into clumps that correspond to the actions of target-machine instructions.
10	Control Flow Analysis	Analyze the sequence of instructions into a *control flow graph* that shows all the possible flows of control the program might follow when it executes.
10	Dataflow Analysis	Gather information about the flow of information through variables of the program; for example, *liveness analysis* calculates the places where each program variable holds a still-needed value (is *live*).
11	Register Allocation	Choose a register to hold each of the variables and temporary values used by the program; variables not live at the same time can share the same register.
12	Code Emission	Replace the temporary names in each machine instruction with machine registers.

TABLE 1.2. Description of compiler phases.